PERFORMANCE MANAGEMENT OF EVENT PROCESSING SYSTEMS

CHUNHUI LI

DOCTOR OF PHILOSOPHY

SCHOOL OF ENGINEERING AND APPLIED SCIENCE
ASTON UNIVERSITY
SEPTEMBER 2013

ASTON UNIVERSITY

PERFORMANCE MANAGEMENT OF EVENT PROCESSING SYSTEMS

CHUNHUI LI

DOCTOR OF PHILOSOPHY 2013

THESIS SUMMARY

This thesis is a study of performance management of Complex Event Processing (CEP) systems. Since CEP systems have distinct characteristics from other well-studied computer systems such as batch and online transaction processing systems and database-centric applications, these characteristics introduce new challenges and opportunities to the performance management for CEP systems.

Methodologies used in benchmarking CEP systems in many performance studies focus on scaling the load injection, but not considering the impact of the functional capabilities of CEP systems. This thesis proposes the approach of evaluating the performance of CEP engines' functional behaviours on events and develops a benchmark platform for CEP systems: CEPBen. The CEPBen benchmark platform is developed to explore the fundamental functional performance of event processing systems: filtering, transformation and event pattern detection. It is also designed to provide a flexible environment for exploring new metrics and influential factors for CEP systems and evaluating the performance of CEP systems.

Studies on factors and new metrics are carried out using the CEPBen benchmark platform on Esper. Different measurement points of response time in performance management of CEP systems are discussed and response time of targeted event is proposed to be used as a metric for quality of service evaluation combining with the traditional response time in CEP systems. Maximum query load as a capacity indicator regarding to the complexity of queries and number of live objects in memory as a performance indicator regarding to the memory management are proposed in performance management of CEP systems. Query depth is studied as a performance factor that influences CEP system performance.

KEYWORDS: Complex event processing; Performance management; Benchmark; Response time; Throughput; Complexity of queries; Garbage collection.

To my parents

DECLARATION

I declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research. Parts of this work will be published as:

C. Li and R. Berry. CEPBen: A Benchmark for Complex Event Processing Systems. Performance Characterization and Benchmarking, Springer International Publishing, 2014, 8391, p.p. 125-142. *Part of Chapter 3 and 4.*

Chunhui Li

ACKNOWLEDGMENTS

Doing a PhD is a long journey, full of stories, which make me excited and stressed, laugh and cry. Completion of my PhD gives me the most satisfaction that I have never had before. I'm very grateful for this life experience.

Enormous thanks are due to my supervisor, Prof. Robert Berry, who has patiently advised me and supported me throughout my PhD studies, with great care and knowledge. I am also thankful to him for insightful guidance in improving my work. Nothing in this short paragraph can express my gratitude.

I have received help and support from many friends and colleagues during my PhD studies. I regret that I cannot mention them all by name here. I am very thankful to my friends Mr. Markus Niederlechner and Dr. Muddasser Alam, my colleague Dr. Shahzad Mumtaz, for the invaluable discussion, reviews and suggestions during my PhD. I'm very thankful to Dr. Diar Nasiev for his expertise and great help in mathematical problems. I'm also very thankful to Dr. Andrea Nini for the advice in improving the academic writing and his reviews of my thesis. Many thanks are due to Dr. Shi Su, Dr. Yang Yang and Dr. Guchun Zhang for the enjoyable time we had. Specially many thanks are due to Dr. Xueting Wang, Dr. Meng Song, Miss Yingying Cai and Miss Xi He for their companionship and patience especially when I experienced hard time.

Finally, I am grateful to my family: my parents, brother, grandparents and our extended family members for their great support and care in my life. It would have been impossible without their unremitting support that I could achieve this milestone of a doctoral degree.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

In this chapter, we will give an introduction of complex event processing systems and their performance management. We will also describe our contributions and outline the rest of the thesis. We use both event processing and complex event processing interchangeably to refer to the term "complex event processing".

1.1 INTRODUCTION OF COMPLEX EVENT PROCESSING

Over the last decades, we have witnessed some great developments in computer systems. Computer systems are now used in almost every imaginable field from science to day-to-day activities of our lives. Although the basic purposes of these computer systems are the same, i.e., to automate and increase the ease of tasks, they differ in the computation and the amount of information they can handle. For example, computer systems for commercial companies, banks, and governments handle and process a much larger amount of data compared to other computer systems. These computer systems are sometimes referred to as large scale or global computer systems that process large amount of data. This tremendous increase to process and handle large quantities of information poses scalability challenges for large scale of systems.

Nowadays, ongoing technologies bring us to the big data era. Data are generated from various sources, including social networking and media, mobile devices, internet transactions and networked devices and sensors. For example, Facebook has 699 million daily active users on average and 1.15 billion monthly active users in June 2013, according to reports on Facebook Newsroom[1]. Each Facebook update from every user creates new data. More than 5 billion people are calling, texting, tweeting and browsing on mobile phones worldwide. Akamai analyzes 75 million events daily to improve targeting advertisements[2]. Walmart handles more than

1 Facebook Newsroom: http://newsroom.fb.com/Key-Facts

2 Akamai: http://www.asterdata.com/resources/assets/cs_Aster_Data_4.0_Akamai.pdf

1 million customer transactions every hour[3]. Appliances such as sensors are widely applied both in daily life and research for monitoring the environment. Sensor networks may consist of different types of sensors such as thermal, visual, acoustic and radar. They can monitor ambient conditions, including temperature, humidity, pressure, noise level, movement of certain objects [3]. Sensors produce large volumes of data continuously over time. The large volume of data generated in every field challenges existing information technology (IT) architecture to process information effectively and efficiently.

Complex Event Processing (CEP) is a set of techniques and tools that provides new opportunities to tackle the challenges of large volume of data. CEP can be applied broadly because information systems are all driven by events. It defines and utilizes relationships between events and helps users to understand the ongoing process in their system. CEP is also flexible. It allows users to specify the events that are interesting to them at any time. Various kinds of events can be specified and monitored simultaneously.[4]

An event is defined as follows [1]:

> "An occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. The word event is also used to mean a programming entity that represents such an occurrence in a computer system."

The occurrence of an event is significant because it might affect other events and actions. Events are produced by an event producer, which might be an application, a service, a business process, a sensor or a transmitter. Events are consumed by event consumers, which receive the events subject to patterns. Event processing refers to computing that performs operations on events.

Figure 1.1 shows the general architecture of an event processing system. In event processing applications, events could be generated by hardware such as sensors, industrial equipment, or software applications. An event consumer could be a hardware component which acts physically according to the events. It could be a software, which may take records of the events it receive. The intermediary event processing involves event routing, event filtering, event transformation, events patterns matching and so on.

3 The Economist News: `http://www.economist.com/node/15557443`

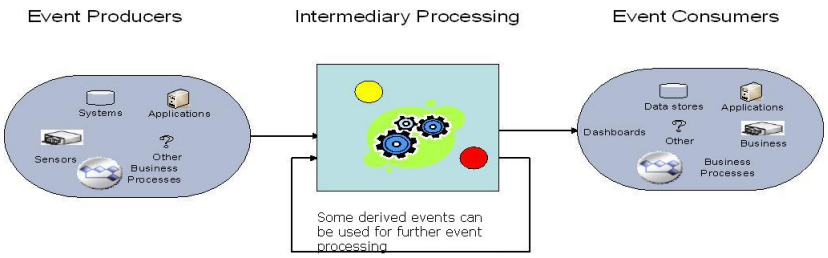


Figure 1.1: The general architecture of event processing [1]

CEP detects multiple independent events by patterns in various dimensions of interest and derives new events, which summarize, represent or denote a set of other events [2]. For example, in a temperature sensor network, the system detects 10 adjacent sensors reporting temperatures higher than 30 centigrade, and then generates an alert event to the end users. This process is complex event processing. This alert event is a derived event.

Event processing has been widely applied. Applications of event processing can be categorized into the following areas[1]:

- **Observation**. Event processing is used for monitoring a system or detecting certain behaviour in a process and generating in-time alerts. For example, an event-driven approach is applied in wireless sensor networks in the health-care environment which is to monitor patients' vital signs as much as information requested is by doctor or nurse stations [5].

- **Information dissemination**. Information dissemination applications include applications such as stock, sports and news tickers, tourist, travel and traffic information systems, and emergency notification systems. Event processing could help in delivering the right information to the right event consumer at the right time. A typical example is the work done by Bacon et al. [6]. Based on the Transport Information Monitoring Environment project, they design and develop a software to distribute, process and store sensor data in real-time. They aim to make data widely available to policy makers, application developers and citizens.

- **Dynamic operational behaviour**. Event processing is used to drive the actions of a system dynamically, to react to incoming events.

For example, an online trading system matches buy requests and sell requests in an action. Based on the trading policy, order information and risk evaluation according to the trading history, the system carries on the process of trades. Another example is subsequence matching in stock systems, mentioned in the paper by Wu H. et al. [7]. Online event-driven subsequence matching is developed and demonstrated to perform better, compared with the approach of fixed time period search.

- **Active diagnostics**. Event processing can help to diagnose problems based on the events that have occurred in systems. Altman et al. present an event-driven, distributed system-level diagnosis algorithm to enhance the performance of fault diagnosis in parallel multi-computers [8].

- **Predictive processing**. Event processing can be used to predict the incoming events by analyzing the events that have already happened. Due to the development of distributed computing systems, higher level of automation in systems management tasks, such as diagnosis and prediction based on real-time streams of computer events, setting alarms and monitoring, is required. For example, Sahoo et al. present a proactive prediction and control system for large clusters [9].

Event processing techniques provide a flexible, extensible and natural way to integrate event data into an application. Event processing is very appropriate and practical in many cases. It is well suited for applications which are naturally centred on events, e.g., sensor networks in monitoring environment. An example of this type of applications is the forest fire detection application with millions of sensor nodes set in forests and bio-complexity mapping of the environment. It is also well fitted to be used in real-time business systems which have high requirements on timeliness, scalability and handling large volume of data.

1.2 PERFORMANCE MANAGEMENT

After a brief introduction to complex event processing systems, in this section we discuss the general performance management of computer

systems and new challenges and opportunities that event processing systems face in performance management.

1.2.1 *Performance of event processing systems*

Performance is a fundamental concern for all users of computer systems. Achieving acceptable performance at reasonable cost is an important requirement. Various approaches and frameworks to enhance the performance of computer systems were proposed in literature, e.g., [10, 11, 12, 13, 14, 15, 16]. For a computer, performance is measured by the amount of useful work accomplished, associated with the time and resources used. For the users of a computer system, performance is measured against expectations and negotiated levels of service. For providers of computer applications, performance is achieved using following mechanism:

- **Application design**: Designing approaches and developing methodologies to utilize hardware resources and to handle increasing work loads effectively.
- **Sizing and configuration**: Determining the type of hardware needed to support performance goals.
- **Parameter tuning**: Applying tools and techniques to set configurable parameters to achieve the best performance.
- **Performance monitoring**: Instrumentation and determining what resources are being used and the level of service the system is providing and users are experiencing.
- **Troubleshooting**: When the system cannot meet the performance requirement, the application is able to diagnose the problems.

Performance management is necessary to optimize utilization of computer systems and to ensure that goals are consistently being met in an effective and efficient manner. It always starts with performance objectives. With clear definition of objectives, performance management involves four essential parts: performance monitoring, performance evaluation, performance tuning and performance prediction.

How to validate that CEP systems satisfy the performance requirements of the applications and how to manage the performance of CEP

systems are the challenges that CEP system vendors and practitioners currently face [17]. Understanding the factors that affect the system performance is very important for performance management of CEP systems.

In the next section, we will present new challenges and opportunities that complex event processing systems face.

1.2.2 *New Challenges in the Performance Management of Complex Event Processing Systems*

CEP systems share certain common performance requirements, but there are wide variabilities in different applications of event processing: e.g., some need millisecond response time while others can accept minute or hour or day response time; some handle several events per day while there are applications which process thousands of millions events every day. As a consequence of the variability, performance management of event processing systems is of great interest and challenge [18].

Event processing systems have distinct characteristics from other well-studied computer systems such as batch and online transaction processing systems and database-centric applications. These characteristics introduce new challenges and opportunities to the performance management for event processing systems, as described below:

- **Independence of event producers and event consumers**

  In an event processing system, event producers and event consumers are independent. Event producers are not aware of the complexity of the processing and applications which are going to consume or interpret these events. Although events have specified event attributes (e.g. header attributes and payload attributes [1]) and content at design time, there is no tight link between event producers and event consumers when events are generated. Figure 1.2[4] shows an overview of an event processing system and the independence of event producers and event consumers. Event consumers capture events they are interested in from the event "cloud", which is a collection of events sent by event producers. New events, event consumers and event sources can be added independently.

4 Courtesy of International Business Machines Corporation, © (2010) International Business Machines Corporation

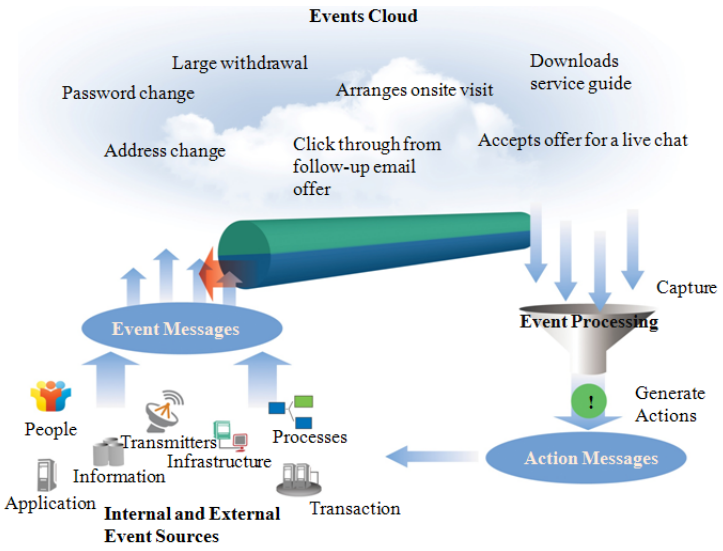


Figure 1.2: Independence of event producers and event consumers [2]

In determining the performance of a computer system, throughput is the usual measure. Throughput measures the number of jobs that are processed in a unit of time in a system. However, because of the independence between event producers and event consumers, a CEP system's input and output are very flexible and dynamic. From a performance perspective, the capability to deliver a satisfying quantity of output events is as important as the capability to process a satisfying quantity of input events.

- **Asynchrony**

  In an event processing system, due to the independence of event producers and event consumers, there is no cause and effect link between the interests of end users (event consumers) and the occurrence of an event. The events that occur might push actions out to end users; might initiate a work flow in an information processing system; might sit indefinitely in a complex event processing condition; or might have no effect at all. This is quite different from a transaction processing system, which uses request-response interactions. The end user initiates a request (e.g., a credit card purchase) and the system returns the information asked for as the response to the request.

  Figure 1.3 shows the differences in response time between transaction processing and event processing. The response time for a

transaction processing system is the period from the time that the request is sent to the time that the response to the request is received by the user. For an event processing system, the response to a request is made after subsequent events' occurrence (event 1, 2 and 3 as shown in the figure). If we apply the definition of response time in transaction processing in complex event processing, the response time would be from the time when event 3 is detected to the time that the action is generated responding to the related events. But for an end user in an event processing system, the time for finishing a single task from the start (e.g., the time from the occurrence of event 1 to the time that the action is taken) is more considerable. How to define the start of timing for response time becomes very important.

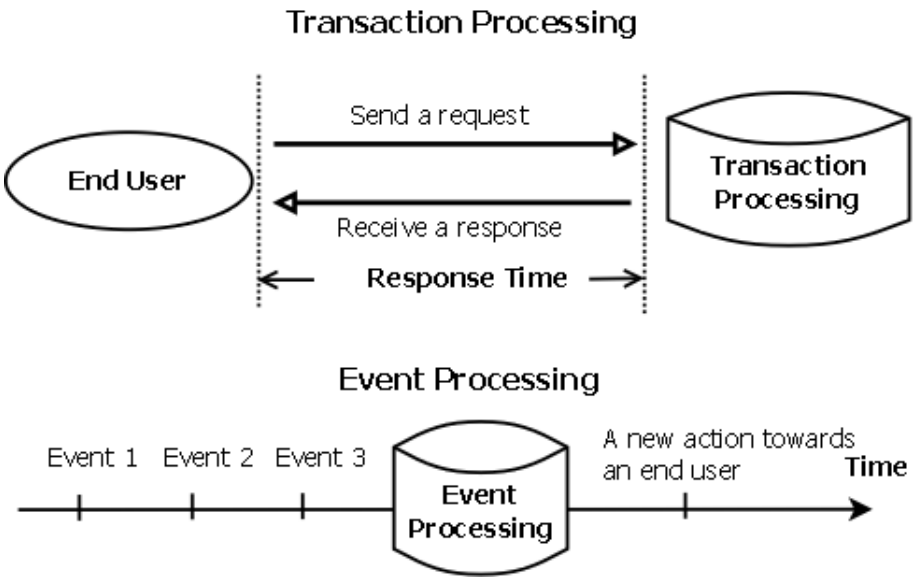


Figure 1.3: Comparison between transaction systems and event processing systems on response time

From a performance perspective, defining the start of a task in event processing systems varies among applications. They might be even different from the different types of events in the same system. This brings challenges to effectively and precisely evaluating the system performance of CEP systems.

- **Timeliness**

Many of today's event processing application areas used to be handled as off-line applications, for example, fraud detection on credit cards or other bank transactions. The growing need for businesses to manage in real time has changed this. Recognizing the significance of an event and identifying related responses to the event could provide a business to respond efficiently to new opportunities and competitive threats, to disseminate relevant information in time to the right people, and to enable active diagnosis of problems.

Therefore, from a performance management perspective, the challenge is how to define the performance objectives of timeliness requirements, considering the response time discussed above.

- **Incremental Evolution**

  Event processing systems grow incrementally. New event types can be easily added into the systems and new complex event analysis can be introduced, which the systems are required to handle in all system layers. Systems can also grow physically in size and geographic dispersion as new event sources are introduced. Take the Birmingham highways project for example. There are 90,000 lampposts in Birmingham highways network. More than one sensor can be set on each lamppost. The throughput can scale up by increasing the intervals of events, sensor nodes and the event types at hundreds of thousands of times.

  Although there are plenty of traditional software engineering approaches that support incremental growth, e.g. iterated waterfall, spiral, develop-refine, extreme programming, event processing is distinct not only because of the independence feature of event producers and event consumers, but also because of the ease of integrating events from new sources with existing event streams.

  This growth model introduces challenges in designing a workload model to simulate the real event input streams containing dynamically changing numbers and types of events in performance evaluation of CEP systems.

- **Statefulness**

  In a stateless event processing system, the processing of one event does not influence the way that the system processes any sub-

sequent events. Stateless event processing systems are easier to manage and to optimize. However, more complex applications are of our interest, in which state management is a requirement. Stateful event processing processes events in the way that is influenced by more than one event input. Many event processing systems are inherently stateful. More complex analysis requires substantial information be maintained regarding prior event occurrence. For example, an application looking for trends in user behaviours will need to record information about prior activity of those users for comparison to current and future activity. [1]

State management can present restrictions to performance management activities such as recoverability, workload management (e.g., routing of events) and load balancing across event processing systems. Restoring the state of a system to its exact value before a failure occurred is very important. States must be maintained consistently. These activities related to states affect event flows so that states have effect on the workload of the systems [19]. It is also resource consuming. Therefore, detecting bottlenecks caused by management is critical and challenging in the performance management of CEP systems.

- **Complexity of queries**

  In event processing systems, deployed queries define the function that the systems perform on input events. Due to the involvement of large volume and different types of data and different interests on the input events from users, queries deployed in event processing systems can be complex. The complexity of queries and the number of such queries all have impact on performance.

  From a performance management perspective, the challenge is to find out how the complexity of queries influences performance, which helps to understand and optimize a CEP system's behaviour.

In the rest of the thesis, we will present our work addressing the above challenges on performance management of complex event processing systems.

1.3 CONTRIBUTIONS AND THE OUTLINE

Our work focuses on the performance management of complex event processing systems. The major contributions of this work are as follows:

- A review of the state of art of the performance management in complex event processing systems is presented.
- A benchmark platform named CEPBen aiming to provide a testbed is developed to explore the important performance factors and new performance metrics for complex event processing systems. The capability of this benchmark is demonstrated by implementing it on the Esper event processing platform[5].
- Based on the CEPBen framework, new factors and metrics are explored for performance management of complex event processing systems. The performance impact of the complexity of queries and memory management on a CEP system is studied. New metrics including response time of targeted event, maximum query load, and the ratio of live objects in the heap are proposed for the performance management of CEP systems

The rest of the thesis is organized as follows: Chapter 2 reviews the state of art in performance management of transaction processing systems and event processing systems. Chapter 3 introduces the design of the CEPBen framework that is developed as a testbed for performance management of complex event processing systems. The workload model, performance metrics and factors, and design of tests are described. Chapter 4 presents a framework that is designed and applied for implementing the CEPBen benchmark. Following that, details of implementing the framework on Esper and preliminary experiments on Esper are presented. Chapter 5 proposes and investigates new factors and metrics. At last, Chapter 6 summarizes the thesis and discusses potential future work.

5 Esper is a component for complex event processing, available for Java as Esper, and for .NET as NEsper. `http://esper.codehaus.org/`

2

PERFORMANCE MANAGEMENT OF TRANSACTION PROCESSING SYSTEMS AND EVENT PROCESSING SYSTEMS

Computer systems are built to deliver a quality of service that meets the demands of user applications. Decades of development have produced a vast body of knowledge on the problem of computer system performance evaluation. Generally speaking, the performance of a given system can be determined by measuring different metrics such as throughput, turnaround time and availability [20, 21, 22, 23]. Throughput measures the steady-state work capacity of the system. Turnaround time, also called latency or response time, is the delay between the presentation of the input to a system and the receipt of the output from it. Availability measures the likelihood that a system is operating properly at a given moment, or the percentage of time during which it is operating properly.

In this chapter, we will review the research in performance management both in traditional transaction processing systems and event processing systems. The state of the art reveals the gaps and opportunities in performance management in event processing.

2.1 PERFORMANCE MANAGEMENT OF TRANSACTION PROCESSING SYSTEMS

Transaction processing systems share fundamental performance techniques and metrics with CEP systems. A review of the performance management of transaction processing systems provides insight to general performance management issues. In this subsection, we will start with a short introduction to transaction processing systems, then move on to review the performance management of transaction processing systems. We will also highlight the performance methodology and metrics adopted in these systems.

2.1.1 *Introduction to Transaction Processing Systems*

According to Gray J. and Reuter A.[24], a transaction processing (TP) system is defined as follows:

> "A transaction processing system provides tools to ease or automate application programming, execution, and administration. Transaction processing applications typically support a network of devices that submit queries and updates to the application. Based on these inputs, the application maintains a database representing some real-world state. Application responses and outputs typically drive real-world actuators and transducers that alter or control the state. The applications, database and network tend to evolve over several decades. Increasingly, the systems are geographically distributed, heterogeneous (they involve equipment and software from many different vendors), continuously available (there is no scheduled down-time), and have stringent response time requirements."

A transaction processing system includes application generators, operations tools, one or more database systems, networks and operating system software. Within the TP system, there is a core collection of services called TP monitor. It manages and coordinates the flow of transactions through the system.

2.1.2 *Metrics in Transaction Processing Systems*

Hardware performance and software performance are the two aspects in performance evaluation, while software dominates the cost of databases and communications [24]. Many new or improved techniques have been developed to estimate and analyse the performance of a transaction processing system over years. For example, two queueing models are developed to investigate the performance effects of a database system by varying the granularity of locks and the degree of multi-programming [25]. Avritzer and Weyuker present [26] a new way to design an application-independent workload to compare the performance of an existing production platform and a proposed replacement architecture; Weyuker and Vokolos [27] describe their experience of designing test case in a large industrial client/server transaction processing application.

Transaction processing systems are applied to deliver sets of services. There are a number of possible outcomes when a system user requests any of these services. These outcomes represent the quality of the service provided by the system. Performance metrics refer to the criteria used to evaluate and quantify the performance of a system. For each performance study, a set of performance metrics must be chosen based on performance goals and the services provided by the tested system. Moreover, performance metrics should be chosen correctly and not based on biased goals. Manipulating metrics can change the conclusions of a performance study [22]. Incorrect performance metrics can not reflect the level of service provided by the tested systems and can mislead the system developers and users. In addition, the measure of the level of quality of service and the expectation of performance vary among system developers and users.

Metrics for transaction processing systems include throughput, response time, utilization, reliability, and availability [24, 23]. Throughput is defined as the average number of transactions processed per unit of measured time. Response time refers to the system's elapsed time from the point that a request is made by a user or an application to the response is returned to the user or the application. Utilization measures the fraction of time that a computer resource is busy. A system usually is composed of multiple modules. Module reliability measures the time from an initial instant to the next failure event. Reliability is statistically defined as mean-time-to-repair (MTTF). Service interruption is statistically defined as mean-time-to-repair (MTTR). Module availability measures the ratio of service-accomplishment to elapsed time. Therefore, availability is statistically defined as $\frac{MTTF}{MTTF+MTTR}$.

The definitions of commonly used performance metrics need to be modified to suit certain applications. Therefore, definitions of those common metrics in transaction processing systems will vary depending on applications and types of computer systems.

2.1.3 *Benchmarks*

Benchmarks are prototype applications created for one of several reasons:

1. They can be used to demonstrate and test application behaviours against expectations under very controlled conditions;
2. They can be used to explore performance characteristics of an application under varying, but controlled conditions;
3. They can also be used for comparative studies of distinct run-time environments. One example of applying benchmarks in performance evaluation is that Fortier and Michel demonstrated evaluating the performance of four top industrial databases by running a standard benchmark on the test databases [23].

Some standard benchmarks have been defined to represent the workload of different type of problems. These standard benchmarks are introduced by the industry consortium, Transaction Processing Performance Council (TPC)[1]. They are widely accepted and applied for performance measurement in industry. These benchmarks can be found on their website: TPC Benchmark C (TPC-C), TPC Benchmark DS (TPC-DS), TPC Benchmark E (TPC-E), TPC Benchmark H (TPC-H), TPC Virtual Measurement Single System Specification (TPC-VMS), TPC-Pricing, and TPC-Energy. Each of these benchmarks represents the workload of a TP application pattern. We will now introduce these benchmarks briefly.

- TPC-C [28] is a benchmark addressing the workload of on-line transaction processing (OLTP) systems. It compares OLTP performance on various hardware and software configurations. Maximum Qualified Throughput (MQTh) which is the total number of completed transactions per minute, is known as the performance metric. In the specification, a response time (RT) is defined by

  $$RT = T_2 - T_1$$

  where:

  T_1 and T_2 are measured at the Remote Terminal Emulator (RTE) and defined as:

  T_1= timestamp taken before the last character of input data is entered by the emulated user.

  T_2= timestamp taken after the last character of output is received by the emulated terminal.

1 Transaction Processing Performance Council: `www.tpc.org`

- TPC-DS [29] represents the workload of decision support systems. It models scaling and database population, queries and data maintenance and implementation rules. It is claimed that it has been designed to be broadly representative of modern decision support systems. The size of the test database can be scaled up to 100 terabytes (TB). The benchmark has a data generator to generate valid dataset for the tests, and a query generator to generate functional SQL queries following the query template. Data maintenance operations are performed as part of the benchmark execution. These operations including insert, update and delete operations, implemented in SQL consist of processing refresh data.

  TPC-DS defines three primary metrics:

  - The performance metric, which is the effective query throughput of the benchmarked configuration;
  - The price performance metric, which is defined as price/the performance metric;
  - The system availability date required in the benchmark TCP-Pricing [30].

  It also defines the following secondary metrics:

  - Load time, which start from the creation of the tables (required by the schema of the data) or when the first character is read from any of the flat files or when the first character is generated by the data and query generator (whichever happens first);
  - Power test elapsed time, which measures the ability of the system to process a sequence of queries in the least amount of time in a single stream fashion. And the elapsed time of each query in the power test, which is power test elapse time/number of queries;
  - Throughput tests elapsed times, which measures the ability of the system to process the most number of queries in the least amount of time with multiple users and data maintenance activity;

  The power per performance defined in TPC_Energy Benchmark [31].

- TPC-E benchmark simulates the On-Line Transaction Processing (OLTP) workload of a brokerage firm [32]. The focus of the benchmark is the central database that executes transactions related to the firm's customer accounts. It is claimed that the database schema, data population, transactions, and implementation rules have been designed to be broadly representative of modern OLTP systems. The primary performance metric applied in this benchmark is called reported throughput. The value of this metric is based on the Measured Throughput. The Measured Throughput is computed as the total number of Valid Trade-Result Transactions within a chosen period of time during steady state (the Measurement Interval) divided by the duration of the Measurement Interval in seconds.

- TPC-H benchmark [33] consists of a set of business oriented ad-hoc queries and concurrent data modifications. It represents decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and generate answers to critical business questions.

  Two tests are run in measuring the performance in TPC-H benchmark: One is the power test, which measures the raw query execution power of the system when connected with a single active user; the other one is a throughput test, which measures the ability of the system to process the most number of queries in the least amount of time. The performance metric in the benchmark is called the TPC-H Composite Query-per-Hour Performance Metric (QphH@Size). It reflects multiple aspects of the capability of the system to process queries. These aspects include the selected database size against which the queries are executed, the query processing power when queries are submitted by a single stream, and the query throughput when queries are submitted by multiple concurrent users. The TPC-H Price/Performance metric is expressed as $/QphH@Size.

- TPC Virtual Measurement Single System Specification (TPC-VMS) benchmark defines four new benchmarks by adding the methodology and requirements for running and reporting performance metrics for virtualized databases to the TPC-C, TPC-E, TPC-H and TPC-DS Benchmarks. Details can be found in the specification [34].

Apart from the industrial standard benchmarks, numerous domain-specific benchmarks have been developed over the years of development of transaction processing systems. For example, the HyperModel Benchmark [35] and the object operations benchmark [36] (OO1 Benchmark) are designed focusing on the important characteristics of engineering applications such as computer-aided software engineering (CASE) and computer-aided design (CAD); XML Data Management benchmark (XMach-1 Benchmark) [37], XOO7 Benchmark [38], XMark [39], Transaction Processing over XML benchmark (TPoX Benchmark) [40] are designed to evaluate the performance of XML data management system; A comparison study of these XML benchmarks can be found in [40]. Either response time or throughput is adopted in these benchmarks, except TPoX Benchmark apply both of response time and throughput as the performance metrics; APB-1 benchmark [41] by On_Line Analytical Processing (OLAP) Council in online analytical processing domain; The Sequoia 2000 [42] is a benchmark in spacial data management. A collection of further benchmarks for transaction processing systems can be found in [43] and [44].

2.1.4 *Summary*

In this subsection, we introduce transaction processing systems and the general performance metrics in performance management of such systems. We also review the existing benchmarks in transaction processing systems. Existing benchmarks in transaction processing systems are in large numbers. Therefore, only some benchmarks as examples from different domains are mentioned. The review mainly focuses on the standard benchmarks from TPC. TPC has had a significant impact on the industry and expectations around benchmarks. Vendors and end users both benefit from these standard benchmarks.

In the next section, we will review the performance management of complex event processing systems, focusing on the factors and metrics, performance modeling and analysis and techniques applied in improving the performance of CEP systems.

2.2 PERFORMANCE MANAGEMENT OF COMPLEX EVENT PROCESSING SYSTEMS

There are a number of complex event processing systems existing in both industry and academia, e.g., Esper[2], streambase[3], and Nastel AutoPilot[4] from industry, SASE [45], Cayuga [46] and NEXT [47] from academia. There is also much research on performance management of CEP systems in the literature. In this section, we will review the critical work on performance management of CEP systems in the following aspects: performance factors and metrics, and performance modeling and analysis, and existing technique in performance optimization.

2.2.1 *Factors and Performance Metrics*

Passing rigorous performance tests and analysis before starting production is essential for developers and users. The scenarios of applied complex event processing range broadly and have different operational requirements in terms of throughput, response time, type of events, patterns, number of event sources and consumers, scalability, and more. Factors that influence the performance of a CEP system and metrics of the performance are important to be identified when evaluating performance. Common performance metrics of interest are: the expected event notification latency, utilization and message throughput of the various system components [14].

Throughput is a critical metric when judging the ability of systems to handle large amount of data. Mendes et al. take throughput as the metric and conducted a series of tests to compare the performance of three event processing systems [18]. Lakshmanan et al. [48] choose throughput to demonstrate that their novel approach could achieve high scalability especially when the model and network topology change frequently. Throughput measurement is also applied in the paper by Wu et al. [45]. A complex event system, SASE, is developed to address the need of sliding windows and value-based comparisons between events in monitoring applications using radio frequency identification (RFID) technology. Their work focuses on high volume streams and extracting events from

2 Esper: `http://esper.codehaus.org/`
3 Streambase: `www.streambase.com`
4 Nastel Technologies: `http://www.nastel.com/`

large windows. Throughput is used as a performance metric in their performance evaluation. Oracle publishes a white paper on the performance of Oracle Complex Event Processing. The output event rate, average latency, 99.99% latency and absolute max latency are the metrics in the performance tests [49]. In addition, Isoyama et al. evaluate throughput as the performance metric for their scalable context delivery platform in [50].

Latency is the time that a system takes for the output events to emerge after the input event happened. In the paper by Grabs and Lu [17], the authors propose system latency and information latency as metrics for event processing systems to address the challenges of out-of-order arrival events. The system latency is well understood as it is the time that a system takes to produce the response events from the moment that all needed events are detected for generation of the response. The information latency is the delay caused by late-arriving event or waiting for additional input.

Common metrics that are discussed in Section 2.1.2 are eligible to apply in performance measurement of CEP systems. As CEP systems are different from transaction processing systems at several aspects that are discussed in Section 1.2.2, the definitions of common metrics will be different when they are applied in the performance management of CEP systems. Our work will focus on adapting and defining common metrics and exploring new metrics in performance measurement of CEP system.

2.2.2 *Performance Modeling and Analysis*

As opposed to conventional request/reply-based distributed systems, no general approach to performance modeling and evaluation exists for complex event processing [14]. Kounev et al. state that their work of workload characterization and performance modeling was the first regarding distributed event based systems (DEBS). A workload model of a generic DEBS is developed and operational analysis techniques are used to characterize the system traffic and derive an approximation for the mean event delivery latency. Queueing Petri Nets (QPN) are used for accurate performance prediction. This is a pioneering work in perfor-

mance modeling for distributed event based systems, and it evaluates the throughput and event notification latency. [14]

Mendes et al. [51] introduce a framework for performance evaluation of complex event processing systems[5]. Three challenges for benchmarking of complex event processing systems are presented in this publication. Firstly, there is a lack of standards in query languages, data formats, semantics or terminology. This causes difficulties to design the workload and the interface between a benchmark and tested CEP systems. Secondly, complex event processing is applied in multiple domains. Performance requirements vary in different domains. Therefore, designing a benchmark for CEP systems with flexible workload settings is much needed to meet the performance requirements of different domains. Thirdly, the services provided by CEP systems are different. The performance of various CEP systems can be measured by different metrics depending on the services.

Challenges in developing benchmarks for CEP systems are not limited to the three discussed above. Identifying primary factors and secondary factors before designing a benchmark is important. Performance impact from primary factors needs to be quantified, while performance impact from secondary factors are not of interest to performance analysts [22]. However, very little about performance factors of CEP systems has been investigated.

In addition, Mendes et al. conduct a set of experiments by running a number of micro-benchmarks on three different event processing engines to find out what their bottlenecks are [18]. A core set of operations used in most scenarios: windowing, filtering (selection/projection), transformation, sorting/ranking, aggregation/grouping, correlation/enrichment (join), merging (union) and pattern detection are proposed. Following these operational requirements, the designed experiments are conducted and throughput are used as the performance metric. It is the first performance study to compare different CEP engines. However, the authors do not measure the responsiveness of the tested CEP engines. As real time processing is a very important feature of CEP systems, it is critical for CEP systems to generate responses to the requests quickly. The approaches of performance modeling and analysis in this study can be applied in the performance analysis of our work. In addition, the conclu-

5 The BiCEP project: `http://bicep.dei.uc.pt/index.php/Main_Page`.

sion drawn from these studies can be used to verify our proposals and analysis.

2.2.3 *Existing Techniques in Performance Optimization*

Much work to improve the performance of event processing systems in different aspects can be found in the literature, e.g., [52, 53, 54, 48, 45, 55]. Techniques from other areas are also applied to enhance the performance of event processing systems. Control theory is applied in improving scalability of event processing systems. Xu et al. propose a new architecture for event processing [56] with a controller for the flow control and load balancing in inter-event processing. Event processing could be divided into intra-event processing (e.g., filtering) and inter-event processing (e.g. detecting a resource bottlenecks). The intra-event processing is defined as processing on isolated events, while the inter-event processing is defined as processing on related events to build relationships between them.

Artificial intelligence technologies are applied to performance improvement of event processing systems. These applications can be found in several publications, such as [57, 58, 54]. Saboori et al. [58] apply a well-known evolutionary algorithm called Covariance Matrix Adaptation (CMA) in distributed systems to achieve automatic system performance tuning. By applying CMA to configure the parameters of systems, the authors improve the throughput without increasing the response time. The work by Chen et al. [54] proposes adaptive algorithms that are designed and used for project development. The authors provide methods to pre-process structural events for creating unique name-value pairs that can be correlated by event correlation engines. Two algorithms applying Bayesian network and Monte Carlo sampling for CEP are designed and experimented to improve efficiency and accuracy of event materialization under uncertainty [52].

2.2.4 *Summary*

Most work in complex event processing by now focus on either techniques of implementing high-performance event processing or techniques of improving the performance of a event processing system. Our review focuses on the approaches and analysis of performance management

presented in these papers. The approaches and analysis are categorized into performance factors and metrics, performance modeling and analysis, and techniques in performance optimization. It is revealed that factors and metrics in CEP systems are not well studied. Our work on developing a benchmark for CEP systems and exploring influential factors and metrics in CEP systems will meet the gap.

2.3 BENCHMARKS RELATED TO COMPLEX EVENT PROCESSING SYSTEMS

In this section, we will review four benchmarks that are built in different fields related to event processing: BiCEP, a benchmark for event processing systems; Linear Road, a benchmark for stream data management systems; SPECjms2007, a benchmark for message-oriented middleware; BEAST, a benchmark for active databases. In Section 2.3.5, we will discuss their advantages and their shortcomings when applied in event processing systems.

2.3.1 *BiCEP*

BiCEP is a project in event processing of the University of Coimbra to benchmark Complex Event Processing systems (CEP)[6]. The objective is to identify some of the core CEP requirements and develop a synthetic benchmark or a set of benchmarks to allow a comparison of products and algorithms in spite of their architectural and semantic differences [59].

In [59], the gap in current CEP research is presented, that is no agreement upon terminology, semantics, query languages, data formats, APIs, standards or benchmarks. The main considerations for designing a benchmark of CEP is also proposed. And more specifically, the following metrics for a CEP benchmark to assess and other issues affecting performance are described:

- **Sustainable throughput**: the steady-state number of events per unit of time that event processing engine can process while executing queries.

6 BiCEP Research Project: `http://bicep.dei.uc.pt/index.php/Main_Page`

- **Response time**: the time since the last event of some event pattern is fed into the system until the system notifies the event pattern detection
- **Scalability**: the ability to scale-up, speed-up and load-up. The scale-up will be assessed by increasing the system (event producers and consumers) and increasing the load. The speed-up will be assessed by increasing the system and maintaining the load. The load-up will be assessed by maintaining the system but increasing the load.
- **Adaptivity**: the system's ability to adapt its query processing when unpredictable events occur.
- **Computation Sharing**: devising query processing techniques so that different queries can share computation.
- **Similarity search and precision and recall**: this refers to information retrieval metrics, e.g., precision and recall [60].

The Pairs benchmark [61] is developed as the first of the BiCEP Benchmarks, aiming at assessing the ability of CEP engines in processing large amount of events and simultaneous queries and providing quick responses. It simulates an investment firm where a number of analysts interact with an enterprise trading system that is responsible for automating and optimizing the execution of orders in stock markets. The performance metric is defined as: $p_{score} = \frac{load}{99^{th}\, latency}$, which represents overall system performance. The load in the metric refers to the load of a CEP system processes and 99% latency.

FINCoS framework [51, 62] is developed in the BiCEP research project. It contains a set of benchmarking tools for aiding end-users and developers to carry out performance evaluation on diverse CEP systems. The performance metrics built in the framework are throughput and response time. It is a helpful tool for preliminary performance evaluation. However, it lacks flexibility in exploring and measuring other performance metrics.

2.3.2 *Linear Road Benchmark*

The linear road benchmark has been created for Stream Data Management Systems (SDMS) by Arasu et al. [63]. It simulates a toll system for motorways, where tolls are set according to dynamic factors, such as traffic congestion and accident proximity. It is designed to evaluate the performance of the systems to respond to real-time queries in processing high-volume streaming and historical data. The benchmark is implemented on two systems. One is a commercially available relational database system and the other is a pre-release commercialization of Aurora [12].

The researchers identify and propose several challenges to design a benchmark for streaming data. Firstly, the input data should not be purely random but have some semantic validity, because a typical stream presents discrete measurements of a continuous activity. Moreover, consistency is required between the content of a stream and the activity that it presents. Secondly, the typical database benchmark metric of "completion time" is not appropriate because of the presence of continuous queries. Thirdly, the validation of the correct results in the benchmark should have multiple correct answers from the same query because some states are evolving. At last, there is no standard query language for stream management systems yet.

It is claimed that the linear road benchmark meet each of these challenges in the following manner:

- A traffic simulator [64] is used as semantically valid input, instead of a purely random input generator in the linear road benchmark.
- Response time and L-rating (i.e., supported query load) are chosen as the metrics for the benchmark. L-rating that refers to the number of motorways, is a measure of the amount of input that an SDMS can process while meeting response time and correctness constraints.
- Multiple correct answers in response times are attained, because the answers to these queries vary depending on evolving states of the toll system.
- Queries in the system are specified formally in the predicate calculus instead of a stream query language.

2.3.3 *SPECjms2007*

SPECjms2007 [65] (Standard Performance Evaluation Corporation) is a benchmark to provide a standard workload and metrics for measuring and evaluating the performance and scalability of Message-Oriented Middleware (MOM) platforms based on Java Message Service (JMS). It provides a standard workload and performance metrics for competitive product comparisons, as well as a framework for in-depth performance analysis of enterprise messaging platforms.

SPECjms2007 measures the end-to-end performance of all components in the application, including hardware, JMS server software, Java Virtual Machine (JVM) software, database software if used for message persistence, and the system network. SPECjms2007 provides three different workload topologies which correspond to three different modes in which the benchmark can be run: two controlled topologies, horizontal topology and vertical topology and freeform topology. Horizontal topology is scaling in a horizontal direction for increasing the number of JMS Destinations and associated Event Handlers whilst maintaining a fixed throughput per destination. The Vertical topology is the scaling in a vertical direction for increasing the throughput per JMS Destination whilst maintaining a fixed number of those destinations. The freeform topology allows users to define their own topology and scale the workload in an arbitrary manner.

SPECjms2007 can serve different purposes. It is capable to produce and publish standard results for marketing purposes. The capability of tuning and optimizing system platforms and analyzing the performance of certain specific MOM features are attractive to many users. Other users may be interested in using the benchmark for research purposes in academic environments. For example, one might be interested in evaluating the performance and scalability by novel methods and techniques for building high-performance MOM servers (e.g., [66, 67]).

2.3.4 *BEAST*

BEAST (BEnchmark for Active database SysTems) is a benchmark for Active Database Management Systems (ADBMSs) [68]. It is based on the OO7 benchmark, which was built for performance tests of Object-

Oriented Database Management Systems (OODBMS) in 1993 [69][70]. BEAST benchmark can serve the three purposes. Firstly, it can be run by users to compare the performance of multiple ADBMSs. Secondly, it can be used by designers to identify the performance weakness of their systems compared with others. Thirdly, it can be applied to compare the performance of an ADBMS with the performance of a passive database management system, where the active behavior is encoded manually in the applications.

The goals of performance measurement for BEAST is to measure the execution time of the services provided by ADBMS [68]. The tested parameter in the measurement is active behaviour. The active behaviour is composed of three phases: event detection, rule management and rule execution. BEAST proposes a collection of tests focus on specific subtasks of the three phases. Event detection is the recognition of the occurrence of specific events of interest, and it includes primitive events and composite events. The accuracy of the detection is an influential aspect in performance. Rule management affects the performance by identifying and retrieving corresponding rules to help to determine if a primitive event will be used in a composite event. Rule execution identifies the condition and actions that have to be executed after event occurrences and the execution of such identification. BEAST measures rule retrieval time, but does not consider rule definition and rule storage in ADBMS for rule management. The performance tests are listed in the Table 2.1.

The papers by Geppert et al. [13] present further performance tests using the BEAST benchmark. The performance tests are held by applying BEAST to four different object-oriented ADBMS which are ACtive Object Oriented Database (ACOOD) management system [71] , Native Active Object System (NAOS) [72], Ode [73] and SAMOS [74]. The performance measurements demonstrate achievements in the area of active database technology, and also show trade-offs between performance and functionality.

2.3.5 *Discussion*

In this section, we review four significant benchmarks related to event processing. We summarise the main goals and technologies of these benchmarks in the following table (Table 2.3).

Table 2.1: BEAST tests

	BEAST tests
For primitive events detection	Detection of value modification, message sending, transaction events and a set of different primitive events
For composite events detection	Detection of a sequence of primitive events, the non-occurrence of an event within a transaction (negative event), the repeated occurrence of a primitive event, a sequence of events that are in turn composite, a conjunction of method events sent to the same receiver object and a conjunction of events raised by the same transaction
For rule execution	The execution of single rules: Different approaches linking and processing condition and action parts are compared in the execution of single rules group
	The execution of multiple rules: Different strategies to execute multiple rules are tested by the second subgroup.

Table 2.3: Summary of benchmarks

Benchmarks	**Developers**	**Focus**	**Technologies**
Linear Road	Stanford University, Brandeis University, MIT, and OHSU/OGI	to determine the performance metrics of a stream processing metrics system	Stream Data Management Systems (SDMS)
BEAST	Institute for Information of University Zurich	to identify performing and inefficient components of systems	Active Database Management System (ADMS)
BiCEP	University of Coimbra	to identify the core CEP requirements; to help compare the products and algorithms	modern CEP engines
SPECjms2007	Standard Performance Evaluation Corporation	to evaluate the performance of enterprise message-oriented middleware servers	MOM JMS

Performance metrics are decided by different performance objectives in applications. According to the literature, the performance metrics of benchmarks in event processing vary. In the Linear Road benchmark [63], response time and supported query load are proposed as appropriate metrics for the system. The BEAST benchmark adopts response time as the metric for performance measurement. Other factors are the number of defined events, the fraction of composite events and the number of defined rules, besides the factors for passive DBMS, such as buffer size, database size, etc [68]. In the BiCEP benchmark, sustainable throughput, response time, scalability, adaptivity, computation sharing, and similarity search and precision and recall are the metrics that the authors considered [59]. Details of these metrics have been given in Section 2.3.1.

These four benchmarks have different goals. Linear Road and BEAST benchmarks partly match with the requirement of a test environment for performance evaluation of event processing systems. However, Linear Road is a benchmark focusing on data streaming with a workload of traffic and transportation verticals. It is not sufficient to investigate the statefulness and complexity of queries. BEAST has a series of tests for rule execution and events detection, which none of the other three benchmarks have. However, it is a benchmark of active database systems, and it lacks state operation, which is a very important property of event processing. The BiCEP project focuses on benchmarking event processing system, although it is still under development. The objective of their performance research is different from that in our project. The BiCEP aims to develop a suite of standard benchmarks for CEP systems, while our project focuses on a benchmark which can be used as a testbed to explore the performance metrics and factors of CEP systems. SPECjms2007 is the only industry-standard benchmark among those that we explored. It is well designed and developed, but as it provides standard workload to measure performance and scalability of JMS-based MOM platforms, it is not to be used to explore performance management in statefulness and complexity of event processing languages. Therefore, to explore the performance of event processing systems, we need to create a testbed for our research.

2.4 SUMMARY

In this chapter, we give a brief review of performance management of transaction processing systems and a comprehensive review of the state of the art in performance management of event processing systems. Performance management of transaction processing systems has been studied well. Numerous benchmarks have been developed for performance management of applications from various domains. Therefore, it is impossible to cover all the benchmarks. Only those widely accepted and deployed TPC benchmarks are reviewed and some other benchmarks from different domains are briefly listed.

In comparison, performance management of CEP systems is not studied as well as transaction processing systems. Definitions of performance metrics for CEP systems are barely found in literature. Several benchmarks in stream data management systems, message-oriented middleware and active database management systems, which are related to CEP systems are reviewed. In addition, one benchmark for CEP systems has been identified but it is currently under development.

It is found that performance research in event processing mostly stays within different application areas instead of benchmarks for general complex event processing, and lacks standards. Most performance measurements and evaluation in event processing systems are done in demonstrating new event processing techniques in different publications. Limited work is done focusing on fundamental aspects in performance of event processing systems, for example, benchmarks for CEP systems. New performance metrics and factors distinguished from other types of systems are not well explored in CEP systems. Methodologies of performance management for CEP systems is not studied much as well.

3

BENCHMARKING COMPLEX EVENT PROCESSING SYSTEMS

Performance management ensures that performance goals and the promised level of service are consistently being met in an effective and efficient manner. Benchmarks are often created for exploring performance characteristics of an application under varying but controlled conditions. However, following the discussion in Chapter 2, no existing benchmark is found sufficient for comparing the performance and exploring the performance evaluation of event processing systems. Therefore, a benchmark platform for performance evaluation on CEP systems is developed.

In this chapter, we will focus on the benchmark platform that is built to explore performance characteristics of complex event processing systems by evaluating the functionalities of such systems.

3.1 INTRODUCTION TO THE CEPBEN BENCHMARK PLATFORM

In this section we will introduce the motivation and the goals to develop the CEPBen benchmark platform and the tested system for this benchmark platform.

3.1.1 *Motivation and Goals of the Benchmark Platform*

Complex event processing excels at in processing large amount of data and responding in timely fashion. As society demand faster reactions to changing conditions, CEP systems are means to meet this demand. Systems in many domains have benefited from event processing technologies, e.g., active diagnostics, real-time operational decision, predictive processing, observation systems and information dissemination.

Complex event processing engines provides three main functional capabilities: filtering, transformation and event patterns detection. A filter operation takes an event input and decides whether this event is to be selected for further processing. A transformation operation takes one or

more input events and generates different output events that are based on them. Event patterns are templates specifying one of more combinations of events. An event pattern detection operation detects such templates [1].

Performance management ensures that performance goals and the promised level of service are consistently being met in an effective and efficient manner. Some performance reports can be found from various CEP venders, e.g. Oracle [49, 75], Esper [76] and StreamBase [77]. Their methodologies in benchmarking CEP systems focus on scaling the load injection, but do not consider the impact of the functional capabilities of a CEP system. The functional capabilities of a CEP system are critical, because they are the foundation of the service provided by the system. We propose an approach of evaluating the performance of CEP engines' functional behaviours on events and create the benchmark Platform for CEP systems: CEPBen.

CEPBen is developed to explore the fundamental functional performance of event processing systems: filtering, transformation and event pattern detection. It is designed to provide a flexible environment which is capable of classifying the performance of CEP systems and exploring new metrics for CEP systems and influential factors in evaluating the performance of CEP systems.

The CEPBen benchmark Platform has the following features of flexibility.

- It is able to present a varied workload to meet the requirements of different performance tests. Users can create events with their desired event properties, batch size, batch frequency.
- By setting the number of query statements and the depth of query statements, the benchmark platform presents varied degrees of application query complexity for investigating the system behaviours.

These features will help to explore a range of factors influencing the performance of CEP systems and a range of metrics to better show that performance.

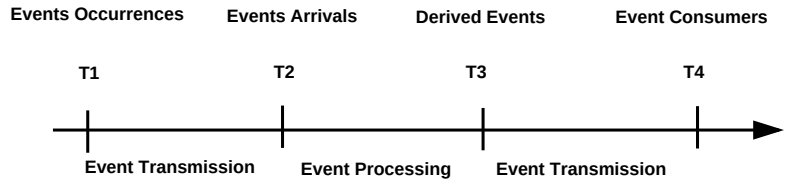


Figure 3.1: The system behavior of an event processing system

3.1.2 *The Tested Systems*

CEP delivers high-speed processing of events, identifying the meaningful events according to defined rules, and taking subsequent action in real time. The goal of this benchmark platform is to measure the efficiency of CEP engines' functional behaviours.

Figure 3.1 illustrates the timeline of events in a CEP system and the system behaviours. Events occur at T_1, and transmit to the front end of the event processing engine at T_2. After being processed, derived events are generated at T_3 and sent back into the CEP engine for further processing or transmitted to event consumers. Event consumers receive the derived events at T_4.

Event transmission and event processing are crucial for the performance of a CEP system. Event sources and event processing engines influence event processing:

- Event sources influence the system by their schema, time, causality, aggregation and input rate;
- While event processing engines drive the whole system to provide satisfactory services in detecting events and their patterns and generating necessary messages for actions.

On the other hand, event transmission is much depending on networks, because event sources and event consumers usually have distributed features. Thus, the performance of networks plays an important role in such scenarios. Since the main focus is on the event processing behaviours in our benchmark platform, the performance of networks is not tested in this platform.

3.2 WORKLOAD DESIGN

In this section, we will firstly review the workloads in existing benchmarks related to CEP systems. Then we will introduce the workload design for the CEPBen. At last, we will evaluate the workload design.

3.2.1 *Workload in Complex Event Processing Systems*

A workload is the amount of work that a system has to perform in a given time. The workload is the most crucial part in performance evaluation [22]. Existing benchmarks have different workloads:

The Pairs benchmark [61] is a domain-specific benchmark. The input of Pairs is stock market data. Its workload consists of sets of five strategies which define operations in a tested system in the application scenario.

The workload in linear road benchmark is generated by the MIT Traffic Simulator (MITSIM) [64]. The simulator generates the stream data and historic data for the benchmark. The stream data consists of the information about position reports, historical query quests for account balances, daily expenditures and travel time estimation. The historic data prior to the start of the simulation need to be maintained by the system for answering historical query requests.

Aiming to provide a flexible framework for performance analysis of JMS servers, SPECjms2007 offers a configurable workload. It provides a list of workload configuration parameters that can be set by users. The workload can be scaled in three workload topologies: horizontal, vertical and free form, all of which have been introduced in in the Section 2.3.3.

Different from other benchmarks above, BEAST does not propose a typical application. Therefore, BEAST workload is defined to test the basic ADBMS functionality: event detection, rule management and rule execution. Details about these benchmarks are reviewed in the Section 2.3.

It is found that most of the workloads are designed in typical application scenarios, except the BEAST workload. Similarly, our benchmark platform focus on CEP engine's functionalities, and no typical application of CEP systems is suggested in CEPBen. In the next section, we will describe the workload design for the CEPBen benchmark platform.

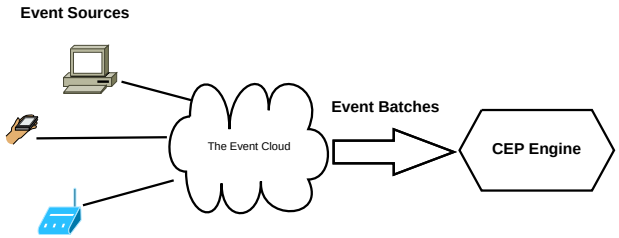


Figure 3.2: The workload model of CEPBen

3.2.2 *Workload Design in CEPBen*

To simplify the abstraction of the workload for performance tests, a batched model for the event workload of CEP systems is designed in CEPBen. The workload consists of event batches of variable size with varying interval times (depending on desired load). Figure 3.2 shows that events from various event sources form the event cloud [78] and arrive in the CEP engine in batches.

The workload can scale in the following dimensions: 1) The number of event batches; 2) The number of events in a batch; 3) The Interval times between two event batches. The interval times in a workload are arbitrarily distributed generally. Moreover, the average time of the intervals can be set according to test scenarios. Short intervals mean heavier workload in a time unit for the system, while long intervals mean less workload in a time unit.

By setting the number of events in each batch and the interval times between two event batches, the workload can vary greatly. This feature also ensures the representativeness of various range of loads in real CEP applications. Users can create their specific event loads according to their application scenarios following the workload model. Because the benchmark platform tests the fundamental functionality of CEP engines, no typical application is proposed in CEPBen for CEP systems.

3.2.3 *Discussion on the Workload Design*

According to Jain [22], four major aspects are considered in designing the workload in performance evaluation of computer systems: the services exercised by the workload, the level of detail, representativeness, and timeliness. The service exercised by the workload should be as complete as possible. The level of detail refers to the details in reproducing

the requests for the services that the system provides, e.g., most frequent request, frequency of request types, time-stamped sequence of requests. A test workload should be representative the workload of the real application. Timeliness refers to that a test workload should follow the users' usage pattern in a timely fashion.

The proposed workload model has rich flexibility to help users to address the major considerations in defining the suitable test workload when using the CEPBen. CEP applications vary in different domains, therefore, workloads for applications vary as well. The workload model is representative for general loads in CEP systems. It can be applied in applications where input events generated at various rate are processed. Users are able to configure the number of events in a event batch and the time intervals between event batches accordingly. The details of input events, e.g. event types and values of event attributes, can be defined by users too. Therefore, the level of details of the workload is decided by the users of the benchmark platform. However, timeliness of usage pattern in a system is not considered in this model, because users' usage pattern does not have much impact on the workload in a CEP system. As event producers and event consumers (users of CEP systems services) are independent in a CEP system, event producers are not aware of the complexity of the processing and applications which are going to consume or interpret these events. On the other hand, event consumers capture events which they are interested in from the output of CEP engines.

3.3 SELECTION OF METRICS

For each performance study, a set of performance metrics must be chosen. Generally, if the system performs the service correctly, its performance is measured by the time taken to perform the service, the rate at which the service is performed and the resources consumed while performing the service. These three metrics are also called responsiveness, productivity and utilization metrics respectively. Response time is the measurement of the responsiveness; Throughput is the measurement of the productivity. The percentage of time that the resources in the tested system are busy for the given load level is the measurement of utilization [22].

Throughput, response time and utilization are commonly applied metrics in measuring information systems. Therefore, these three metrics are selected as the basic metrics of CEPBen. In this section, we will define the measurement of these selected metrics.

3.3.1 *Throughput*

Throughput can be categorized into input throughput and output throughput in light of the independence feature of event producers and event consumers in CEP systems [1]. Suppose that a workload with n event instances is sent into the event processing engine and m output events are generated after the event input are processed. The input throughput is measured by:

$$Input\,Throughput = \frac{The\ Number\ of\ Input\ Events\ in\ the\ Period}{T_{Nth} - T_{1st-input}} \tag{3.1}$$

$$Output\,Throughput = \frac{The\,Number\,of\,Output\,Events\,in\,the\,Period}{T_{Mth} - T_{1st-output}} \tag{3.2}$$

$T_{1st-input}$ and T_{Nth} are the start time and the end time of the sample period for measuring the input throughput after the system reaches steady state. $T_{1st-output}$ and T_{Mth} are the start time and the end time of the sample period for measuring the output throughput after the system reaches steady state.

3.3.2 *Response Time*

Response time is one of the performance metrics in the CEPBen. The response time measurement is defined in the following manner.

When the CEP engine perform filtering, response time is measured as follows: Before each event arrives at the event processing engine, the event is labelled with the system time T_{in}. When the event is selected according to the selection queries by the event processing engine, it is labelled with the current system time T_{out}. The responsive behaviour of the system is measured by the metric of response time:

$$T_{response} = T_{out} - T_{in} \tag{3.3}$$

More than one event are involved in a transformation operation or an event pattern detection in complex event processing. An event arrives in the CEP engine at system time T_{in}. A CEP engine performs event processing on events E_1, E_2,..., E_n, which arrive in the CEP engine in time series ($T_{n-in} > ... > T_{2-in} > T_{1-in}$). The new output event that is triggered by these events is generated at system time T_{out}. The response time of transformation behaviour is defined as:

$$T_{response} = T_{out} - T_{n-in} \tag{3.4}$$

3.3.3 *Utilization*

The utilization is measured by the utilization of the memory resource and CPU resource of the computer where the tested CEP engine is run. The utilization of a resource is measured as the fraction of time when the resource is in busy servicing requests. The utilization of a resource is defined as:

$$U = \frac{Busy\ Time}{Total\ Elapsed\ Time} \tag{3.5}$$

Alternatively, in a busy CEP system which has continuous input event streams, the utilization are the percentages of usage of memory and CPU resources that are measured over a certain period during the busy time as follows:

$$U = \frac{Average\ of\ Used\ Resource}{Total\ Alocated\ Resource\ for\ services} \tag{3.6}$$

3.4 BENCHMARK PLATFORM DESIGN

The goal of complex event processing is to identify meaningful events and respond to them as quickly as possible. Three main functionalities are necessary to perform the event processing: filtering, transformation and event pattern detection. These functionalities are performed according to queries registered in the CEP engines.

3.4.1 *Tests for Filtering*

Filtering is a fundamental functionality of a CEP engine. It can be found in many event processing applications [1]. For example, applications that involve event producers which produce large numbers of events, might need to filter out irrelevant events, such as sensor networks and news feeds. Some applications might have multiple event consumers, so these applications need to perform filtering on the events that are sent to each event consumer.

A filter operation takes an event instance as input and decides whether that instance is to be selected for further processing. In CEPBen, a group of tests that focuses on filtering function of event processing systems is designed and conducted. A load of queries representing filtering operations are created and registered in the tested CEP engine for the performance tests.

3.4.2 *Tests for Transformation*

According to the definition by Etzion and Niblett[1], transformation in event processing can be categorized into the following types: translation, composition, aggregation, enriching, splitting and projection. A translation operation takes a single event as input and generate a derived event following a pre-defined derivation formula. A composition operation takes groups of events from two input streams, looks for matches by a matching criterion and generates derived events based on the matched events. An aggregation takes a collection of events as input and creates a single derived event according to a defined function over input events. An enriching operation takes a single input event and creates a derived event which includes the attributes from the original event, possibly with modified values, and can include additional attributes. A splitting operation takes a single input event and creates a collection of events which can be clones of the original event or events that contain a subset of the attributes of the original event. A projection takes a single event and creates a single derived event containing a subset of the attribute of the input event.

This group of tests focuses on transformation function of event processing systems. Query statements of transformation that cover all types

of transformation for the event processing engine are created and registered in the tested CEP engine. These operation will perform on events in one input stream and different input streams.

3.4.3 *Tests for Detecting Event Patterns*

Pattern detection allows users to look for a specific collection of events and the relationship between them. It helps to understand the meanings of events. Pattern detection operation performs a pattern matching function on one or more input events, and emits one or more derived events when a specified pattern is detected. This group of tests focuses on the system behaviour of detecting event patterns. A load of queries of detecting event patterns are created and registered in the tested CEP engine for the performance tests.

3.4.4 *Factors*

It is important to identify the factors which make a significant impact on the performance of the tested systems. In our benchmark platform, the following factors are identified for the performance of a CEP system:

- The workload.

  Heavy workload with relative information to the system queries challenges a CEP system's capability to respond to a large amount of events.

- The query load.

  Query load is the number of query statements in different test groups. Handling large sets of query statements efficiently is a challenge for CEP engines. As CEP engines take time to process events against each query, it is expected that a larger number of query statements increases the total processing time. On the other hand, processing a large number of query statements consumes more resources of the computer, which slows down the CEP system.

- The number of events that a query statement involves.

Among three main functionalities, transformation produces composite events which are formed based on different number of events. Similarly, an event pattern detection operation can involve various number of events as well. To process events against statements with number of events, the CEP engine need to catch and hold the required events temporarily, which is resource-consuming.

- The garbage collection.

  The benefits of garbage collection over explicit memory management in software engineering are widely accepted. However, the garbage-collection pause that happens when the garbage is collected can be unpredictable. The garbage collector must suspend the execution of the application to ensure the integrity of the object trees when it performs garbage collection. Therefore, garbage collection has direct impact on the performance of a CEP engine.

- The machine configuration where the event processing engine is run.

  The performance of a CEP system relies on the hardware configuration of the machine on which the CEP system runs.

3.5 SUMMARY

In this Chapter, we introduce a benchmark platform for complex event processing systems: CEPBen. This benchmark platform targets at the performance of fundamental functionalities of CEP engine. The workload design, selected metrics and benchmark tests, as well as the factors that are considered to be important for the performance of a CEP system are described.

This benchmark platform is very flexible. On one hand, it can be applied to CEP engines for the performance evaluation of CEP engines. On the other hand, it can be applied to explore novel metrics and influential factors to measure the performance of CEP systems. This will be demonstrated in the later chapters of this thesis.

In the next Chapter, we will introduce the performance-oriented framework that is developed for the performance management of CEP sys-

tems. Based on the framework, CEPBen is implemented on an open-source CEP platform Esper[1].

1 Esper:http://esper.codehaus.org/

4

THE PERFORMANCE-ORIENTED FRAMEWORK AND CEPBEN IMPLEMENTATION

Complex event processing has been increasingly popular in the business operations and planning. Complex event processing products can be found in diverse areas. In the last chapter, we presented CEPBen, a benchmark platform for complex event processing systems. In this chapter, we will focus on the implementation of CEPBen and its performance tests. A performance-oriented framework is designed and presented in the absence of a flexible testbed for performance evaluation of CEP systems . This framework is used in our implementation of CEPBen on Esper complex event processing engine.

4.1 THE PERFORMANCE-ORIENTED FRAMEWORK

In the section, we will introduce the features of the performance-oriented framework and its general architecture.

4.1.1 *Introduction*

Performance management aims at achieving performance objectives and delivering the promised level of service. Benchmarks are developed in order to evaluate performance of CEP systems, which is discussed in Chapter 2 and 3. We can implement and demonstrated the instrumentation, tooling and methodologies that are required for complex event processing systems with an appropriate framework. However, there is a lack of flexible performance tools for performance instrumentation for CEP systems. A framework FINCoS is developed in BiCEP project to provide a flexible and neutral framework which allows users, researchers and engineers quickly run realistic performance tests on event processing platforms without having to code themselves load generation, performance measurement and event conversion routines. However, with built-in per-

formance metrics and measurements, this framework is not sufficient in exploring and testing novel performance metrics.

Therefore, we propose the performance-oriented framework in order to support performance instrumentation and performance tests with a wide range of performance metrics. This framework provides a flexible and scalable testbed environment for evaluating performance of event processing systems, especially in exploring performance metrics, performance instrumentation and performance analysis. It can be adapted to test the performance of various event processing engines and the performance of various algorithms that developed by various researchers.

4.1.2 *Features*

The framework can create event streams according to application scenarios and take other form of data as event resources, i.e., Comma-Separated Values (CSV) files; The input layer and output layer of this framework offer interfaces as measurement points for performance measurement, which allows users to implement and test the performance of their particular interests. The query module is designed to be configured according to users' applications and to help users understand the effect of complexity of queries on the system performance.

4.2 THE DESIGN OF THE FRAMEWORK

In this section, we will introduce the framework that is created for supporting performance evaluation of event processing system. This framework is implemented in JAVA. The implementation details can be found in the appendix.

Figure 1 shows the architecture of the framework. It has the following components: events generator, input and output layer, event processing engine, query module, data collection, real time performance monitor and off-line performance analysis.

As shown in the Figure 4.1, this benchmark platform is composed of the following components: the Events Generator, the Input Layer, the Output Layer, the Event Processing Engine, the Query Module, the Events Consumers and the Performance Analysis Module. The Event Processing Engine is the core component for an event processing system.

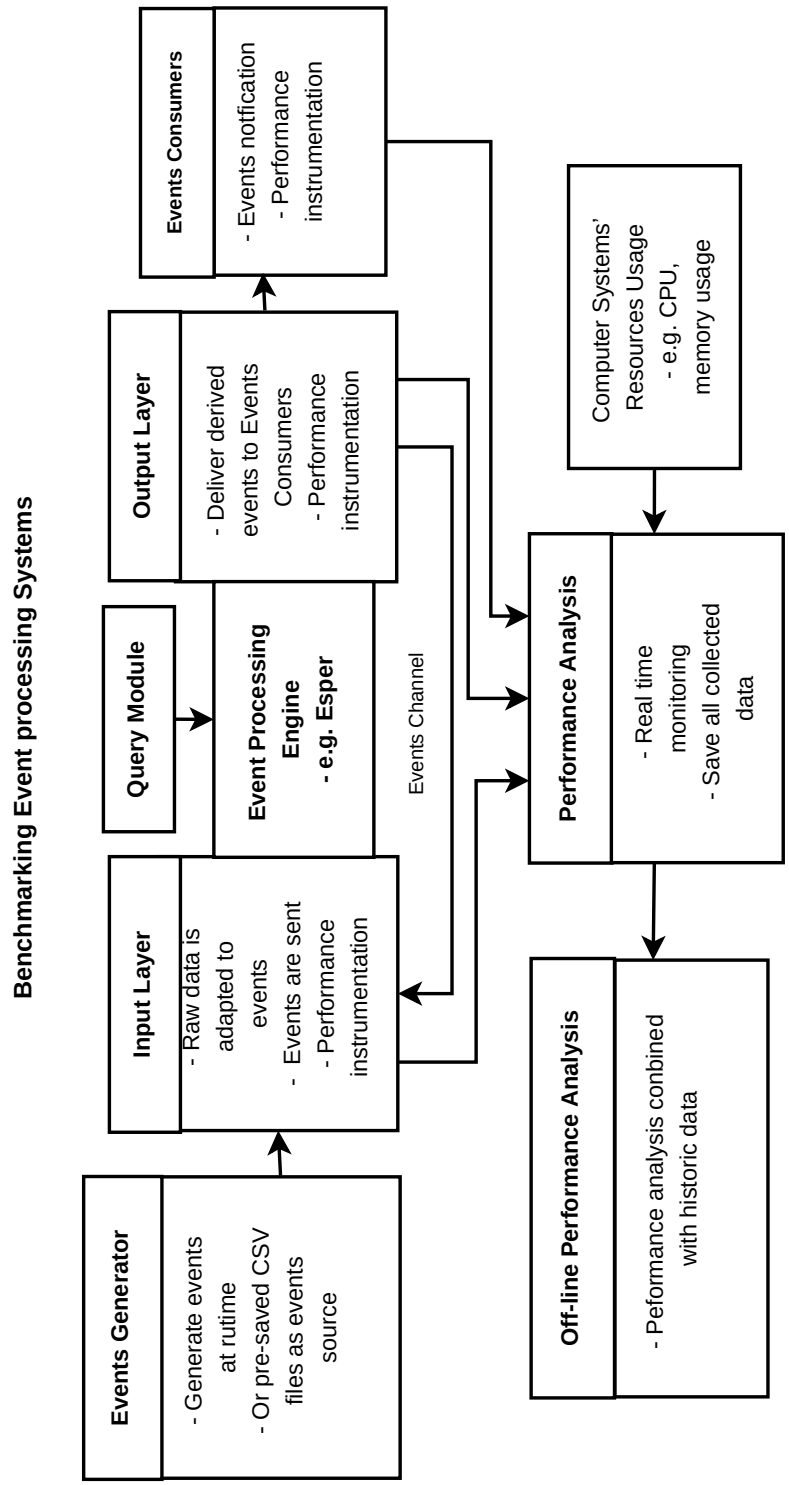


Figure 4.1: The architecture of the framework

Event processing engines from different vendors can be applied here. The Events Generator is the event source for the event processing benchmark platform. The Input Layer is the connector of the Events Generator and the Event Processing Engine. This layer adapts the events that are generated in the Events Generator and sends them to the Event Processing Engine. The Output Layer connects the Event Processing Engine and the Events Consumers, which delivers the derived events to the Events Consumers. The Query Module contains queries that are registered in the Event Processing Engine according to the interests of events consumers. Performance analysis of the system is built in the Performance Analysis module which can produce real-time performance monitoring, as well as the resources of computer systems. An off-line performance analysis program is developed for the analysis combining with historic performance data.

4.2.1 *Events Generator*

Complex event processing can be widely applied in event-driven applications. The events rate and types of events in applications vary accordingly. This leads to difficulties to create a workload to represent all types of workloads, especially in a benchmark platform of event processing systems. An event generator is built for generating the workloads according to workload designs.

The event generator is for generating events. It can be set to two different modes: generating events at runtime or reading events from saved CSV files. To overcome concerns about computer resources, e.g., memory consumption, CPU consumption, the event generator can be installed on remote machines and send events into event processing engines.

- **Generating Events at Runtime**

  Generating events at runtime provides users flexibility to define the events data that they want to use. Users can either make some data according to the scenarios of simulations that they are interested in, or make random data for tests. Events volume, events types and events rate can be well defined and controlled.

- **Converting CSV Files to Events**

Saved CSV files can be used as events feed to event processing systems. This relies on an adapter in the input layer of the system. The adapters in the input layer convert CSV readings to event instances and send them to the event processing engine.

The two modes have their own advantages and disadvantages. Generating data at runtime consumes the memory of the computer which runs the events generator. Therefore, it does not suit for tests with extremely large data amounts, but it provides flexibility of testing event processing applied in various scenarios. Creating events via saved CSV files does not consume as much memory as generating data at runtime. Thus this mode does not pose big challenges on computer resources when creating event streams with large amounts of data. However, it requires users to prepare a file with data which represents the workload. Thus, it suits the users who have some data already.

4.2.2 *Query Module*

A CEP application in the real world usually has a number of queries in different levels of complexity. The query module is designed for generating queries of various levels of complexity. Some complex queries require the information of the history and states of historic events in the system. Therefore, the system needs to allocate resources to keep the history and states of events. This requirement can pose stress on the system and its performance.

Generally, the query module is implemented based on the type of query statements (i.e., selections, joins, windows, and event patterns), the number of different type of query statements and an execution plan. Because there is no standard event processing languages across CEP engines, queries that the query module generates are implemented depending on the event processing language of the event processing engine that users adopt. Query load can be measured here.

4.2.3 *Input Layer*

The input layer is the front end of the event processing engine in the framework. It is critical for the event processing engine and the performance measurement of the event processing system. Adapters for events

that are from various resources and events senders are implemented in this layer. Adapters read the events from socket, which are transmitted from various event sources, unmarshall it and create event instances for these events and send them to the event processing engine. The measurement of input throughput is implemented in this layer. The calculation of input throughput is discussed in Section 3.3.1. The start of measuring response time is implemented in this layer. Response time in CEP systems is discussed in Section 3.3.2.

4.2.4 *Output Layer*

The output layer is for delivering derived events from the event processing engine to events consumers. The derived events can be alerts from detecting certain events patterns, events to change configuration of the system, events to be processed again or events to be deleted. Derived events and events with new configuration information go through the events channel and arrive in the input layer to get processed.

This layer is important for performance measurement of an event processing system. Output throughput is measured in this layer. The end of the measurement of response time is implemented in this layer.

4.2.5 *Event Consumers*

Event consumers are the end users of CEP systems. They are implemented according to the application of CEP systems.

4.2.6 *Complex Event Processing Engine*

Event processing engine is the core component of an event processing system. In this framework, the event processing engine connects to events sources via the input layer, and outputs notification to events consumers via the output layer. As complex event processing becomes increasingly popular, a number of commercial and open source event processing engines are developed and can be tested under this framework.

4.2.7 *Performance Monitoring and Analysis*

Performance monitoring and evaluation needs instrumentation to gather data on executing systems and processes, techniques for data analysis and representation, theories and models which realistically represent computer systems and computer processes. In the framework, the performance monitor can display the performance in a real-time manner. The off-line performance analysis is performed to compare and analyze the performance results that are obtained in performance tests.

4.3 CEPBEN IMPLEMENTATION ON ESPER

In this section, we will introduce the implementation of the CEPBen benchmark platform on Esper based on the performance-oriented framework.

4.3.1 *Introduction to Esper*

Esper CEP engine is an open source CEP platform. It processes data continuously and generate responses in a real-time fashion when the stored queries are matched.

Esper offers an event pattern language to specify expression-based event pattern matching. The event processing engine matches expected sequences of presence, or absence of events, or combinations of events. Esper also offers event stream queries which include windows, aggregation, joining and analysis functions for use with streams of events. These queries are following the Event Processing Language (EPL) syntax.

EPL is designed for similarity with the Structured Query Language (SQL). However, it differs from SQL: EPL is used to represent the different operations needed to structure data in an event stream and to derive data from an event stream.

4.3.2 *Performance Implementation and Settings*

To demonstrate the benchmark platform, we implement CEPBen on Esper complex event processing engine in Java. Event generator is configured to generate four types of events (EventA, EventB, EventC, EventD)

for the workload. By default, each event has event ID and timestamp property. Other string properties for events are drawn from a string pool with a random process. The integer values in properties are generated randomly. The event structure is shown as following:

EventA (String eventID, long timeStamp, String attributeA)
EventB (String eventID, long timeStamp, String attributeB)
EventC (String eventID, long timeStamp, String attributeC1, int attributeC2)
EventD (String eventID, long timeStamp, int attributeD1, String attributeD2)

The workload is composed of 500 event batches, which each batch has 20,000 events. The average of interval times is set to 0.3 seconds.

Input layer and output layer are programed on the Esper engine. Because the behaviours of CEP engines are the main focus, simple event consumers which subscribe text alerts when the events are detected are implemented.

Three types of queries are implemented in the query module: Selection statements are created for tests of filtering functionality; Join statements are created for tests of transformation functionality; Event pattern statements are created for tests of event pattern detection functionality. The Event Processing Language (EPL) is the language of the Esper event processing. EPL is a SQL-like language with SELECT, FROM, WHERE, GROUP BY, HAVING and ORDER BY clauses [79]. EPL queries are created and stored in the engine. Statement examples for the benchmark tests are listed in the following box:

Tests for filtering:

```
select * from EventA where EventA.attributeA = 'red';
```

Test for transformation:

```
select attributeC as averageSize, attributeD as profession
from EventC.std:lastevent() as attributeC,
EventD.std:lastevent() as attributeD
where EventC.eventId=EventD.eventId
```

Tests for event pattern detection:

```
select * from pattern [ every data= EventA
(attributeA='green')
-> (EventB (attributeB = 'UK'))
-> (EventD (attributeD1>3000))
->(EventC(attributeC1<800))
->(EventC(attributeC2>1000))];
```

In the above box, the filtering query catches all the EventA, in which the attributeA has the value "red". The transformation query combines the attributeC from the last EventC event and the attributeD from the last EventD event when the IDs of the EventC event and the EventD event are the same. The pattern detection query in the above box looks for patterns, which an EventA event has attributeA value as "green", an EventB event which has attributeB value as "UK", an EventD event has attributeD1 value larger than 3000, an EventC event has attibuteC1 value smaller than 800 and an EventC event has attributeC2 value larger than 1000 are fed into the system following this sequence.

As the benchmark platform is coded in Java, it is important to clarify Java Virtual Machine (JVM) setting before running the experiments. This benchmark application can take up to 256 megabytes of heap at runtime. The explicit garbage collection is disabled in the experiments.

4.3.3 *Benchmark Results*

Two groups of tests are set up for the performance test. The settings for performance tests are displayed in the Table 4.1: Group 1 is testing the three functional behaviours respectively with 10 query statements; Group 2 is testing the functional behaviours respectively with 100 query statements. Group 1 and Group 2 are designed for revealing the performance effect of different functional behaviours. Each filtering query catches one event when the event matches the specified condition in the query. Each transformation query transforms two events into a new event. Each event pattern query detects five events in a certain sequence. No windows are implemented in the statements in these two test groups.

Table 4.1: Settings of performance test groups

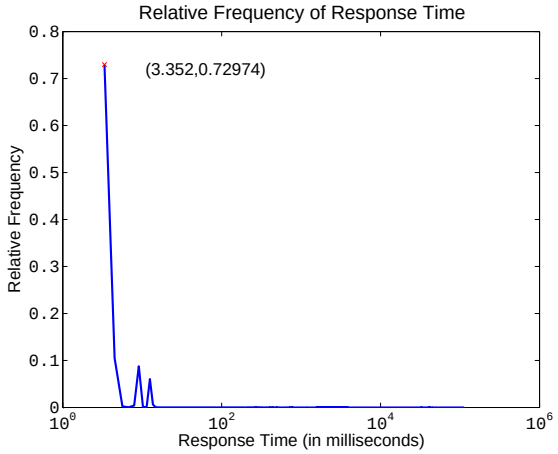
Settings	Group 1	Group 2
Number of queries	10	100
Types of queries	Filtering; Transformation; Pattern Detection	

The input throughput, output throughput, response time and utilization are measured. The input throughput and output throughput are measured as events per second. The averages of input throughput and output throughput are calculated and presented. The response time is measured in milliseconds. Relative frequencies and cumulative distribution of response time are calculated and presented.

4.3.3.1 *Response Time*

TEST GROUP 1 Figure 4.2, 4.3 and 4.4 depict the relative frequency of response time for filtering, transformation and pattern detection in the test Group 1. Each of the three figures contains two graphs: The first graph is the relative frequency of response time plotted on a semi-logarithmic scale showing the shortest response time and the relative frequency of the first bin which contains the shortest response time; the second graph is the relative frequency of response time plotted on a linear scale. In the second graph, the relative frequency distribution is divided into two parts in order to show the curve clearly. The part of the distribution that goes beyond the visible part of the graph is not displayed. The response times of filtering are mainly in the interval between 0 and 15 milliseconds with the most relative frequency above 0.7 (Figure 4.2). The response times of transformation are mainly in the interval between 0 and 40 milliseconds with the highest relative frequency above 0.7 (Figure 4.3). The response times of event pattern detection mainly fall in the interval between 0 to 300 milliseconds with highest relative frequency at about 0.55 (Figure 4.4).

Figure 4.5 presents a cumulative distribution comparison for filtering, transformation and pattern detection in the test Group 1. The part of the distribution that goes beyond the visible part of the graph is not displayed. The cumulative probabilities of response time for filtering and transformation converge to 1 sharply and overlap with each other, while the response time for pattern detection converges to 1 slowly.



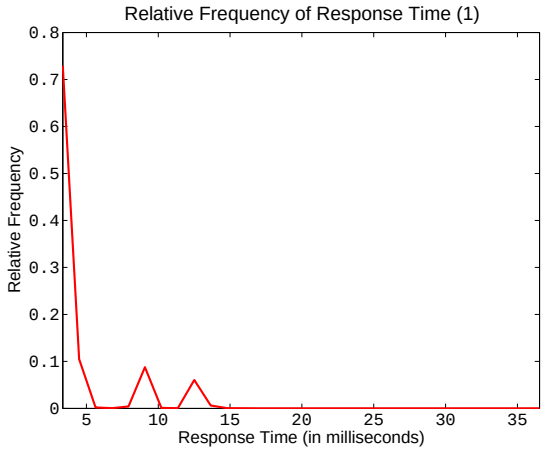
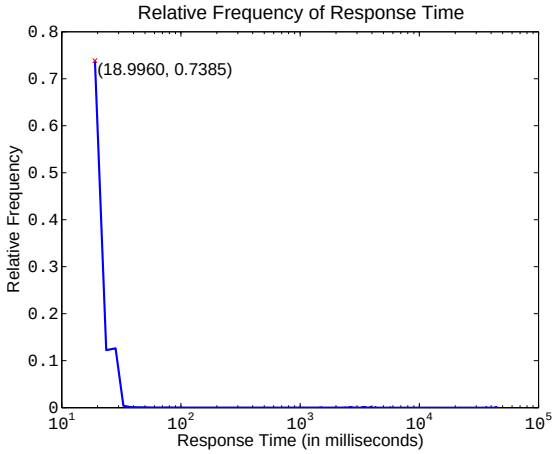


Figure 4.2: Filtering in test Group 1



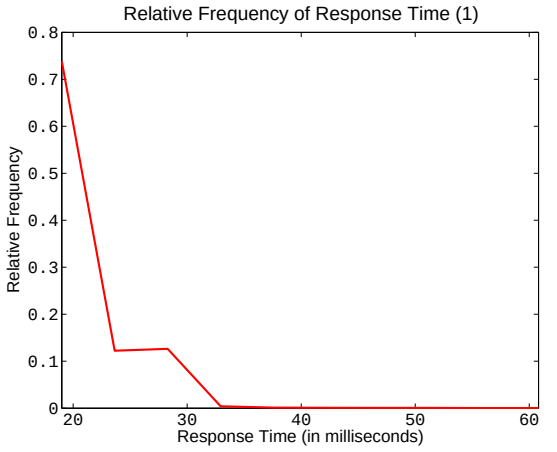
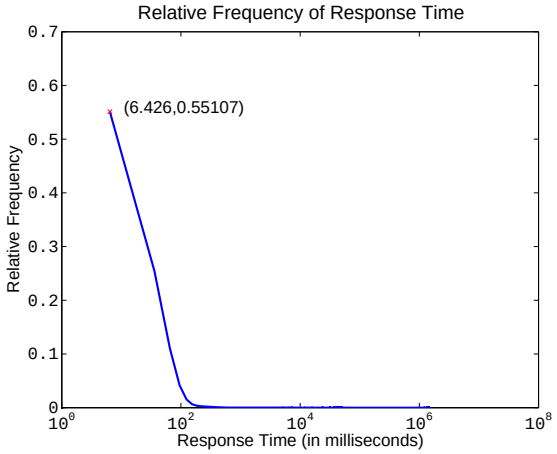


Figure 4.3: Transformation in test Group 1



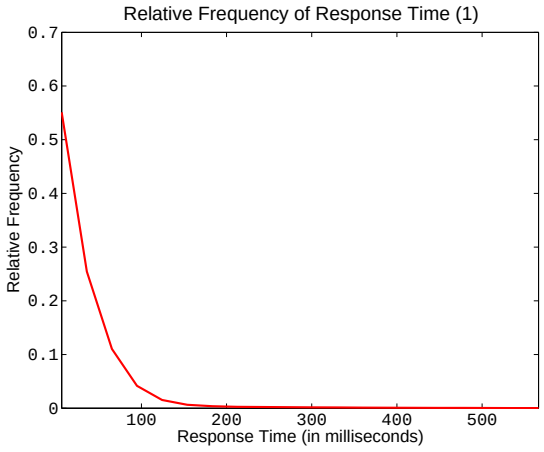


Figure 4.4: Pattern detection in test Group 1

The results of response time in Group 1 reveal that the Esper engine responds faster in filtering than in transformation, and it responds faster in transformation than in event pattern detection.

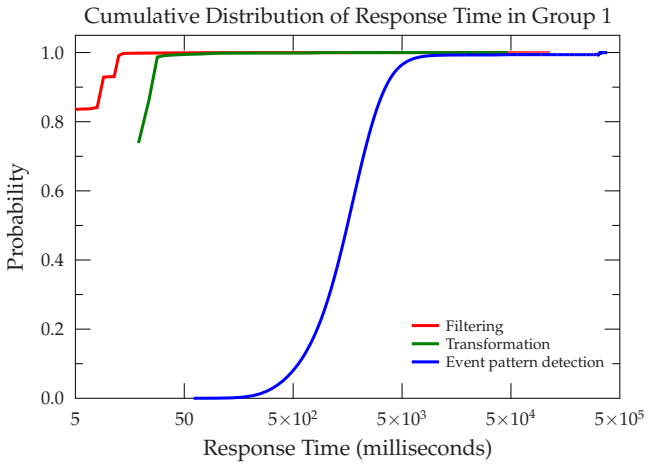


Figure 4.5: The cumulative distribution of filtering, transformation and pattern detection in Group 1

TEST GROUP 2 Figure 4.6, 4.7 and 4.8 illustrate the relative frequency of response time for filtering, transformation and event pattern detection in the test Group 2. Each of the three figures contains two graphs: The first graph is the relative frequency of response time plotted on a semi-logarithmic scale displaying the shortest response time and the relative frequency of the first bin which contains the shortest response time; the second graph is the relative frequency of response time plotted on a linear scale. In the second graph, the relative frequency distribution is divided into two parts in order to show the curve clearly. The relative frequency distribution for filtering are mostly in the interval between 0 and 50 milliseconds (Figure 4.6). The relative frequency distribution for transformation has a higher variability than filtering with a peak of about 0.08 between 160 and 200 milliseconds (Figure 4.7). The relative frequency distribution for event pattern detection has both a higher mean and higher variability (from 0 to 1400 milliseconds) than the other query types (Figure 4.8). The peak relative frequency reaches above 0.25.

Figure 4.9 depicts the cumulative distribution of the response time for filtering, transformation and event pattern detection in the test Group 2. The part of the distribution that goes beyond the visible part of the graph is not displayed. Similarities are found with the cumulative distribution

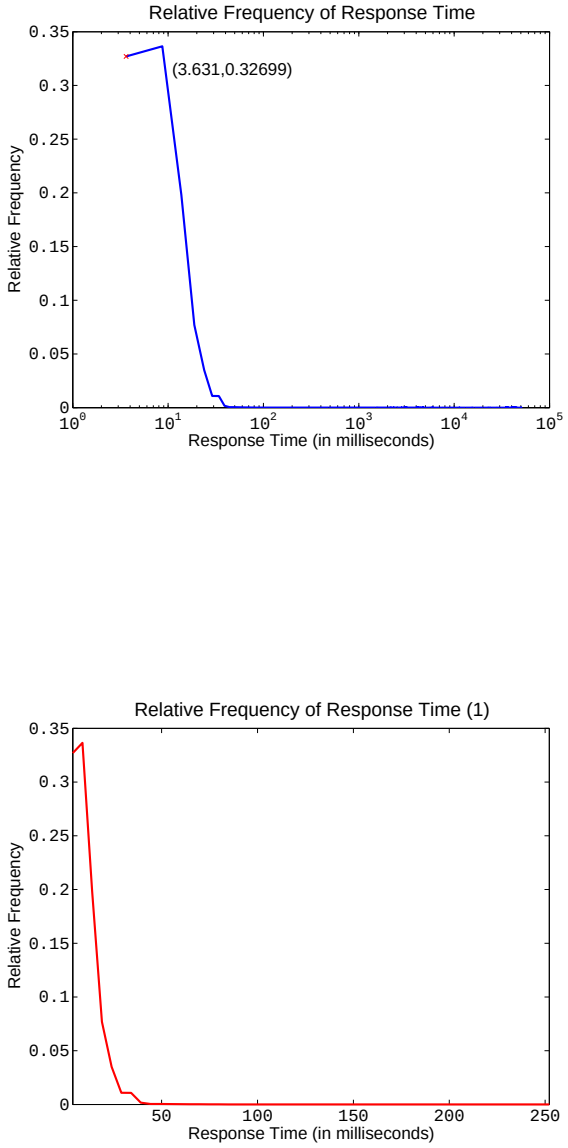
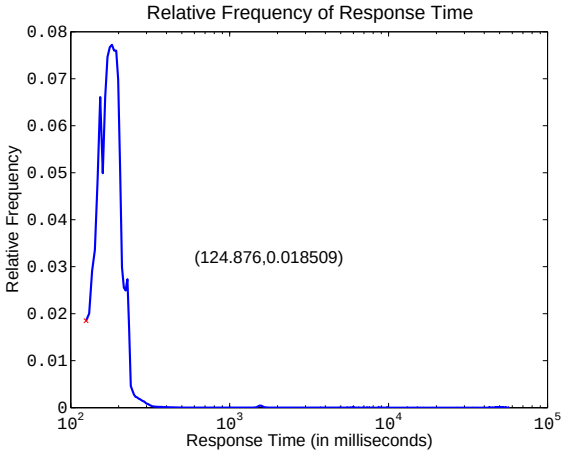


Figure 4.6: Filtering in the test Group 2



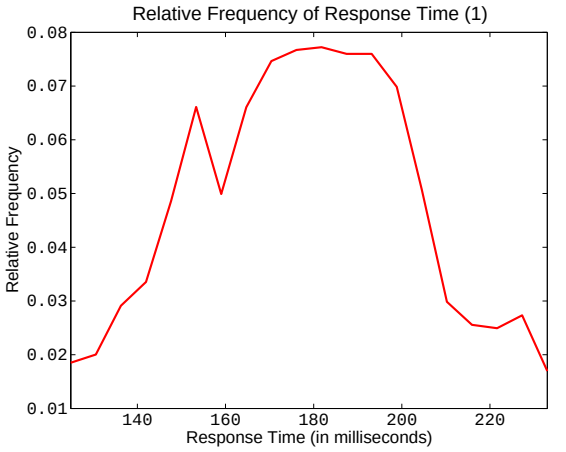
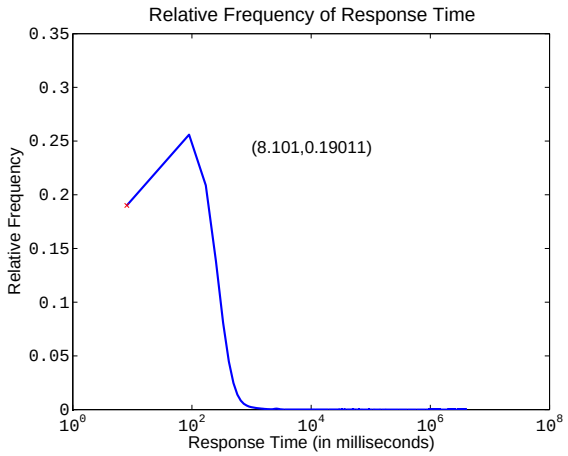


Figure 4.7: Transformation in test Group 2



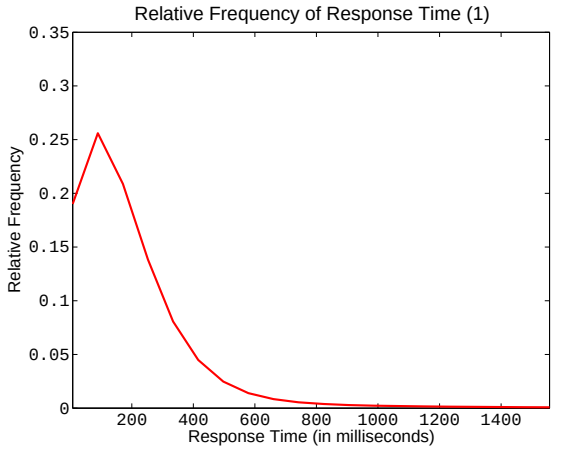


Figure 4.8: Pattern detection in test Group 2

of response time in the test Group 1. The cumulative probabilities of response time for filtering and transformation converge to 1 faster than the response time for event pattern detection.

The results in Group 2 prove the conclusion of the test Group 1 that the Esper engine responds faster in filtering than in transformation, and it responds faster in transformation than in event pattern detection.

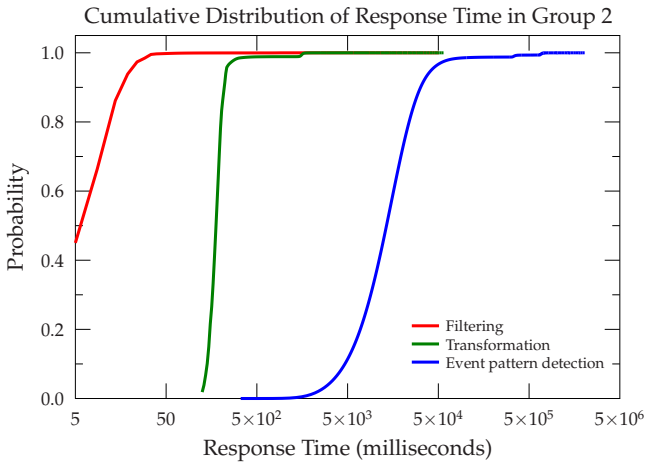


Figure 4.9: The cumulative distribution of filtering, transformation and pattern detection in test Group 2

4.3.3.2 *Input Throughput*

TEST GROUP 1 Figure 4.10 shows the system input throughput in the test Group 1. The input throughput is sampled and measured in each event batch. The system is run for 3 times and the average of the input throughput is calculated and plotted. The input throughput of filtering fluctuates considerably between 140,000 events/second and 210,000 events/second. However, it is much higher than the input throughput of event pattern detection and transformation overall. Noticeably, the input throughput of event pattern detection is higher than the input throughput of transformation.

TEST GROUP 2 Figure 4.11 presents the input throughput of test Group 2. The input throughput is sampled and measured in each event batch. The system is run for 3 times and the average of the input throughput is calculated and plotted. The input throughput of filtering fluctuates between 52,000 and 58,000 events/second, which is much higher than the input throughput of the transformation and event pattern detection in the

same group. However, it is significantly lower than the input throughput of filtering in the Group 1.

4.3.3.3 *Output Throughput*

Outputting events consumes resources of CEP systems. Output load and output throughput in a CEP system can be used to indicate the amount of work processed in the CEP engine with an input workload. Our test environment does not strictly control the amount of output, because values of event properties and values in query statements are generated with a random process. Tables 4.2 and 4.3 present the average output throughput and the average output load in two test groups. Because more queries are deployed in the tests in Group 2, more output events are generated in Group 2, as shown in these two tables.

Comparing the output load and output throughput of three functionalities in Tables 4.2 and 4.3, it is found that the scalability of filtering in Esper engine is the best among the three functionalities. The output throughput and the output load of filtering are both nearly eight times more in Group 2 than in Group 1. The output load of transformation and event pattern detection in Group 2 are nearly ten times heavier than they are in Group 1, while the output throughput of transformation and event pattern detection are nearly twice larger in Group 2 than they are in Group 1.

Table 4.2: The system output throughput and output load in the test Group 1

Functionalities	Output of Group 1	
	Throughput (events/sec)	Load (events)
Filtering	14,431	3,077,108
Transformation	28,644	6,248,000
Event Pattern Detection	3,119	990,876

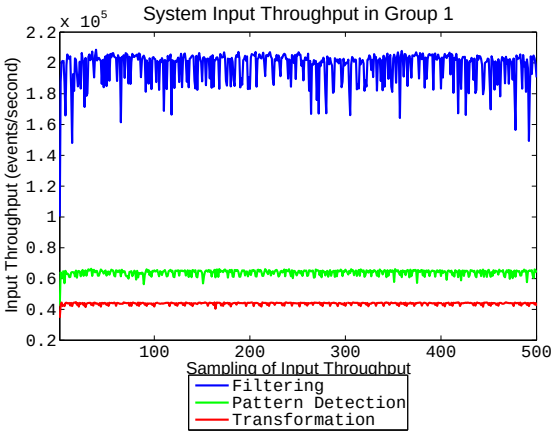


Figure 4.10: The system input throughput in the test Group 1

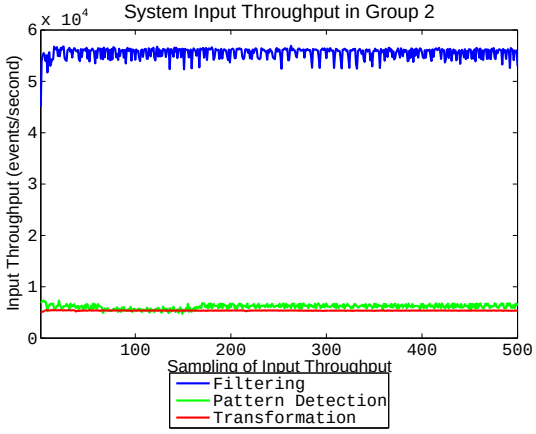


Figure 4.11: The system input throughput in the test Group 2

Table 4.3: The system output throughput and output load in the test Group 2

Functionalities	Output of Group 2	
	Throughput (events/sec)	Load (events)
Filtering	84,366	28,821,616
Transformation	28,644	58,731,250
Event Pattern Detection	5,508	11,086,190

4.3.3.4 *Utilization*

In the experiments, Central Processing Unit (CPU) usage and heap memory that is allocated to the application in JVM are monitored and recorded. VisualVM[1] is used to monitor the utilization features. VisualVM is a tool to monitor and trouble-shoot Java applications. It can profile CPU and memory usage and display memory pool and garbage collection activity, which can be used to spot abnormal trends.

Tables 4.4 and 4.5 summarize the utilization in different tests in the two test groups. The heap size in the experiments is decided by JVM at runtime. The maximum heap size is configured before Java applications are run. In these two groups of tests, the maximum heap size is set to 256 megabytes. The CPU usage fluctuates in a range in each test. Garbage collection is one of the causes. Comparing with Group 1, the CPU usage in group 2 is higher than the group 1 in each type of test. Tests of event pattern detection in Group 2 consume much more memory than other tests in Group 1 and 2.

Table 4.4: The utilization of computer resources in the test Group 1

Functionalities	Test Group 1	
	CPU Usage (%)	Heap Size
Filtering	10% - 20%	16
Transformation	25% - 35%	16
Event Pattern Detection	20% - 100%	256

1 VisualVM is a visual tool integrating several command line Java Development Kit (JDK) tools and lightweight profiling capabilities. `http://visualvm.java.net/`

Table 4.5: The utilization of computer resources in the test Group 2

Functionalities	Test Group 2	
	CPU Usage(%)	Heap Size
Filtering	20% - 35%	16
Transformation	30% -50%	16
Event Pattern Detection	30% -100%	256

4.3.4 *Discussion*

The experiments show that the performance of various types of functionalities in the tested CEP system based on Esper descends in the sequence: filtering functionality, transformation functionality and pattern detection functionality. The tested system has best response time and input throughput in filtering while having high output throughput and consuming the least system resource (i.e., CPU usage and heap usage). Moreover, the tested system has better response time and consumes less system resources in transformation than pattern detection, while having lower input throughput and much higher output throughput. In a summary, the performance with filtering is the best among all functionalities.

A performance study of three CEP systems is done by Mendes et al. [18]. The results is presented to compare three CEP systems with selection and projection queries (filtering functionality), aggregation and window queries (transformation functionality), joins queries (transformation functionality) and pattern matching queries (pattern detection functionality). Throughput is the metric to evaluate the performance. The results show that the level of throughput in system X, system Y and system Z in the tests with selection and projection queries is higher than that with aggregation and window queries and joins queries. The level of throughput with aggregation and window queries and join queries is higher than pattern detection queries. Therefore, our conclusion complies with the conclusion drawn from their experimental results.

In summary, it is tested and found that CEP systems perform better in filtering and transformation than pattern detection. This can be considered and applied in improving the performance of a CEP system.

4.4 GUIDANCE ON THE IMPLEMENTATION OF CEPBEN BENCHMARK FRAMEWORK

In this section, we will provide guidance on the use of the current implementation of CEPBen on Esper to conduct performance tests. We will also discuss what is available to support the reuse of the implementation of CEPBen on other CEP Engines.

4.4.1 *Using the Existing Implementation*

The current implementation of CEPBen on Esper can be used to investigate the performance characteristics of a range of CEP systems. Before a performance test is carried out, the configuration must be specified and this is done by modifying some Java code. There are three components of the framework to be set up: the event generator, the query module and the performance monitoring and analysis.

4.4.1.1 *Configuring the Event Generator*

- Configuring the event streams

  Event streams contain batches of events. The number of events in an event batch and the number of batches that an event stream has are two fields in the Main class in the project. They can be configured in the Main class (i.e., Main.java file).

```
//the number of event batches
static int NUMBER_BATCHES=500;
//the number of events in each batch
static int NUMBER_EVENTS=5000;
```

- Configuring values of event attributes

  Current values for event attributes are all implemented in FieldsForStream class (i.e., FieldsForStream.java file). Users can make changes in this class. For example, the pool of strings used for different attributes and seeds for generating random numbers.

- Configuring types of events in an event stream

  More types of events can be added into an event stream by implementing those event classes and adding them into an event stream.

Configuring types of events in an event stream can be done by adding or removing the line of the code to generate a new event in createEvents method in EventStream class (i.e., EventStream.java file). The code below is an example to add or remove when configuring event streams in EventStream class.

```
//create a random number
long randomA = fields.randomLatency(randomC);
/*create an event of EventA type. An eventA type of //
event has four attributes: event ID, time stamp, the time
when the event is sent into the CEP engine and a random
attribute. */
EventA eventA = new EventA(id, randomA, 0,
fields.getAttributeA());
//add the event into the event stream
stream.add(eventA);
```

4.4.1.2 *Configuring the Query Module*

- Configuring the number of queries deployed in an experiment

  The number of queries deployed in an experiment can be set in the Main class of the project. Different types of queries can be configured with three fields in the Main class shown as follows:

```
//the number of selection query statements
static final int SELECTION = 0;
//the number of transformation query statements
static final int TRANSFORMATION = 0;
//the number of pattern detection query statments
static final int PATTERN = 1;
```

- Add queries with different syntax structures

  Figure 4.12 shows a Unified Modelling Language (UML) class diagram of the query module that is implemented on Esper platform.

  The query generator is implemented to generate queries with a set of fixed syntax structures. For example, the query generator can generate simple queries as follows:

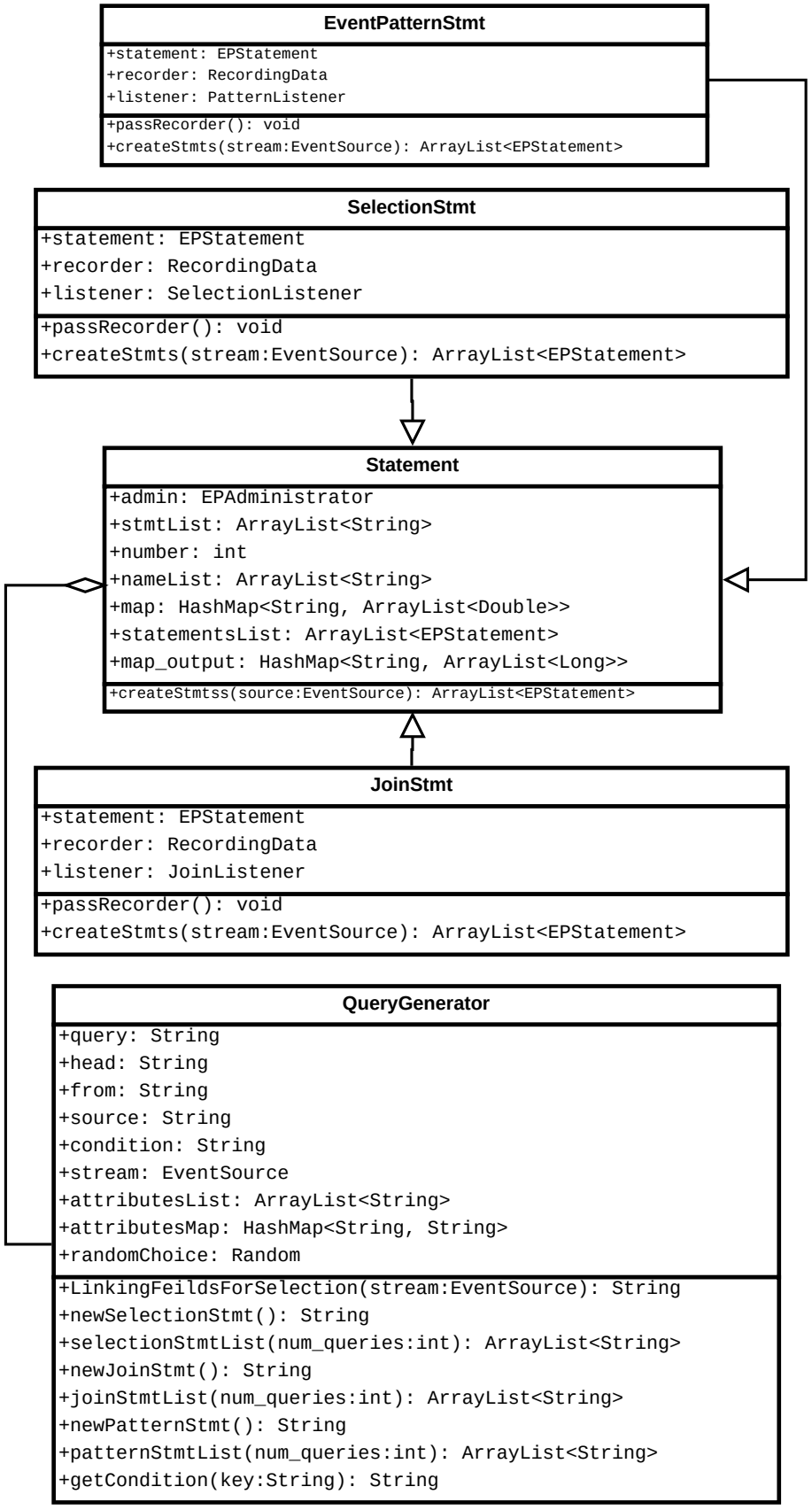


Figure 4.12: UML diagram of the query module

```
select select_list from stream_def [as name] [, stream_def
[as name]] [,...]
[where search_conditions]
```

However, EPL queries can have more variation in syntax structures than the implemented syntax structures, because EPL queries follow the syntax presented in the box below[79]:

```
[annotations]
[expression_declarations]
[context context_name]
[insert into insert_into_def]
select select_list
from stream_def [as name] [, stream_def [as name]] [,...]
[where search_conditions]
[group by grouping_expression_list]
[having grouping_search_conditions]
[output output_specification]
[order by order_by_expression_list]
[limit num_rows]
```

For example, the query generator with current implementation cannot generate queries with some other clauses, e.g., "group by", "order by". To extend the implementation of query syntax structures, users can add new methods in QueryGenerator class (i.e., QueryGenerator.java), to generate queries with different syntax.

4.4.1.3 *Configuring Performance Monitoring and Analysis*

The performance monitoring and analysis can be configured for real-time or off-line mode in the main class. The real-time analysis displays all the response times and throughputs, which are measured while a performance test is running. The performance monitoring and analysis can be turned on or off in the Main class. The framework saves all the performance data when the system is operating in both real-time and off-line analysis mode. These data can be used to analyse the performance after the performance test is finished.

4.4.2 *Implementing CEPBen on other CEP engines*

We introduced the configuration of CEPBen implementation on Esper in the last section. Following this, we will present guidance on the reuse of the implementation of CEPBen on other CEP engines in this section.

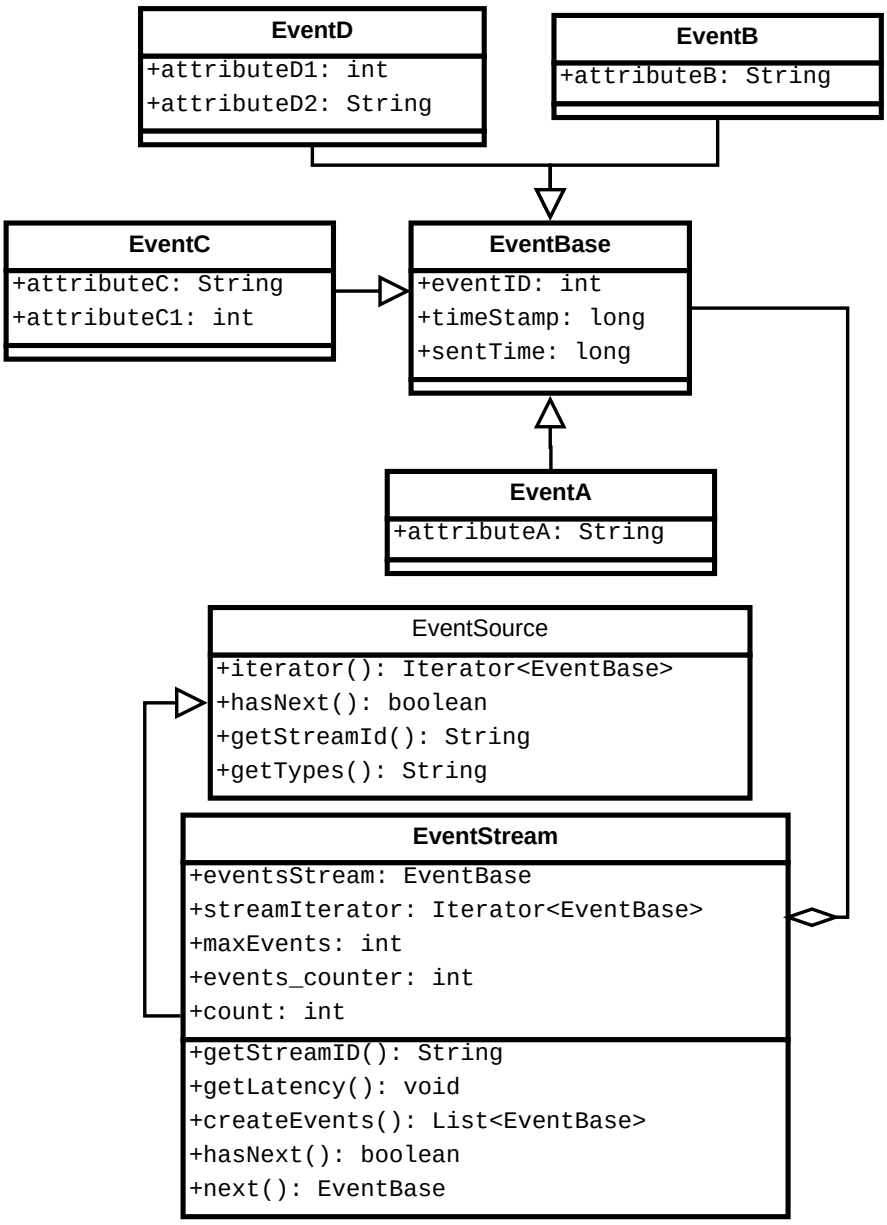


Figure 4.13: UML diagram of the event generator

The event generator component in the framework can be reused and extended. Users can add more event types when needed by modifying

the existing code. Figure 4.13 is a UML class diagram of the event generator.

The event stream is constructed by iteratively adding four events of different types in the implementation (shown in the following box). The values of the attributes in each event are randomly assigned. Therefore, the event stream generated has a fixed structure when there are four types of events: EventA, EventB, EventC, EventD, EventA, EventB, EventC, EventD,.....EventA, EventB, EventC, EventD.

```
for(events_counter=1; events_counter<maxEvents;
events_counter++)
{
    long randomA = fields.randomLatency(randomC);
    EventA eventA = new EventA(id, randomA, 0,
                    fields.getAttributeA());
    stream.add(eventA);
    getLatency(randomC, randomA);
    id=id+1;
    long randomB = fields.randomLatency(randomA);
    EventB eventB = new EventB(id, randomB, 0,
                    fields.getAttributeB());
    stream.add(eventB);
    getLatency(randomA, randomB);
    id=id+1;
    randomC = fields.randomLatency(randomB);
    EventC eventC = new EventC(id, randomC, 0,
                    fields.getAttributeC(),
                    fields.getAttributeC1());
    stream.add(eventC);
    getLatency(randomB, randomC);
    EventD eventD = new EventD(id, randomStamp, 0,
                    fields.getAttributeD1(),
                    fields.getAttributeD2());
    stream.add(eventD);
}
```

The query module cannot be reused with other CEP engines because of the differences between CEP languages. However, the query module can be modified to fit other CEP engines, as the query module produces queries by deconstructing a CEP query to several parts and concatena-

ting them following the format of Esper CEP languange. As shown in Figure 4.12, a query is constructed by four string objects: "head", "from", "source" and "condition". When it is implemented on other CEP engines, different components will be required to construct a query statement. These components of a query statement should be modified accordingly in the methods for generating different types of query statements. For example, to generate selection statements, newSelectionStmt() method has to be modified to meet the CEP language's requirement.

As the input layer links event sources and the deployed CEP engine, and the output layer links the deployed CEP engine and event consumers, the implementation of the two layers should consider event sources, event consumers and the deployed CEP engine in the implementation. Because of the wide variety of event sources, event consumers and CEP engines, the input layer and the output layer must be re-implemented to meet the requirements of the three components of a CEP benchmark.

Performance data is displayed via performance monitoring and analysis module. This module is developed in Java using JFreeChart[2] for displays and opencsv[3] for saving performance data into CSV files. While the performance measurement points discussed in Section 4.2.3 and 4.2.4 are implemented in the input and output layers, the performance monitoring and analysis module is developed and tightly linked to these two layers. As the input layer and the output layer have to be implemented according to the tested engine, this module has to be re-implemented based on performance measurement implementation. The Java libraries, JFreeChart and opencsv, are recommended to CEPBen Users when re-implementing the performance monitoring and analysis module if the benchmark is coded in Java.

4.5 SUMMARY

In this chapter, we present a performance-oriented framework for implementing the CEPBen benchmark platform. The implementation of CEPBen on Esper using the performance-oriented framework is described. Preliminary results of the experiments are obtained and presented as a demonstration of the capability and usability of the CEPBen benchmark platform. Some guidance on how to reuse and extend the benchmark

2 JFreeChart: `http://www.jfree.org/jfreechart/`
3 opencsv: `http://opencsv.sourceforge.net/`

platform using the framework is provided. In the next chapter, we will focus on experiments that are designed to explore the performance factors and metrics. Further discussion on performance management will be presented.

5

METRICS AND FACTORS FOR PERFORMANCE MANAGEMENT OF COMPLEX EVENT PROCESSING SYSTEMS

Performance is an important criterion at every stage in the life cycle of a computer system. Complex event processing systems typically receive continuous input events from various data sources and process these input by queries that continuously and incrementally produce derived events as new input events are detected. Hence, performance management plays an important role in designing and using of a CEP system.

While most of performance studies in CEP focus on scaling load rejection and measuring throughput of CEP systems, e.g., performance reports from Oracle [49, 75], Esper [76] and StreamBase [77], factors and new metrics in performance management of CEP systems have not gained much attention.

Typically, a service level agreement is made to describe the key services that the system providers offer and the quality standards that the system providers and system users have agreed with in terms of service delivery. Key performance metrics as a part of quality standards of services are often stated in the agreement. For example, a service level agreement for a transactional system might have statements like the following:

> "We guarantee that 90% percent of response times will be less than 1 second for all transactions, provided the arrival rate of transactions is less than 10000 per second."

Understanding the factors that affect system performance and selecting appropriate performance metrics are critical to the service level provided by CEP systems. Better knowledge of the system performance metrics and factors helps system developers and users respond to problems and make better evaluation and estimation of their systems' performance. Therefore, it is important to identify those factors which make significant impact on the system performance and explore appropriate metrics that can be applied in CEP systems.

In the discussion in this chapter, the following terms and the example shown in the Figure 5.1 are used to illustrate the performance measurement points in CEP systems.

In complex event processing, each event can have the following header attributes regarding to timestamps: occurrence time and detection time. Their definition can be found in [1] as following:

> OCCURRENCE TIME Occurrence time is the time at which the event occurred in the external system. It is generally set by the application.
>
> DETECTION TIME Detection time is the time at which the event becomes known to the event processing system. It is generated by the CEP system.

Figure 5.1 shows an example of events arriving into a CEP system. Event 1, Event 2 and Event 3 occur at occurrence times T1', T2' and T3' . These events are detected by the CEP system at detection times T1, T2 and T3 respectively in the CEP system. The derived output event is generated as a response in the system at $T_{derived}$. This example will be used later in this chapter to illustrate the performance measurement of CEP systems.

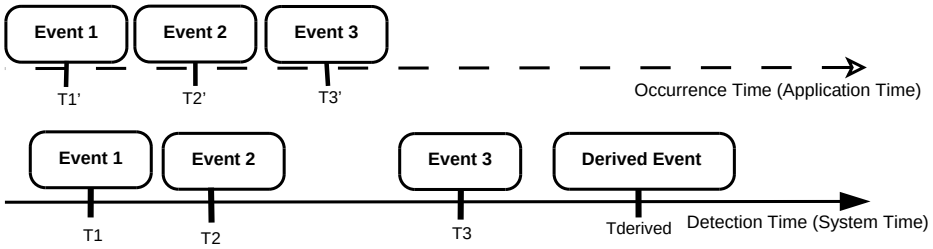


Figure 5.1: An example of events arriving into a CEP system

In this chapter, the studies on the performance factors in CEP systems based on CEPBen will be presented. As a result of the study, three new metrics are proposed and evaluated: response time of targeted event, maximum query load and number of live objects, for performance management of CEP systems. Moreover, query depth as a factor relative to the complexity of queries is studied.

5.1 MEASUREMENT OF RESPONSE TIME

Response time and latency are often interchangeably used to describe the time delay that a system takes for responses to emerge after their

corresponding input events have been detected in the system. Indeed, response time and latency are the key performance metrics in performance evaluation in general.

Interactions between event producers and event consumers in event processing are different from request-response interactions in transaction processing because of the asynchrony feature between event producers and event consumers that is discussed in Chapter 1: There is no cause and effect link between the interests of end users (i.e., event consumers) and the occurrence of an event. In this section, we take a fresh look at response time and propose a different way of measuring response time in performance management of CEP systems.

5.1.1 *Discussion of Response Time*

In performance management, the definition of response time must consider the behaviour, design and architecture of a system in a performance test. Traditionally, response time is measured by the time interval from the time that a user initiates a request to the time that a response to the request is received. Since in complex event processing, an event pattern in event processing usually involves several events to trigger the derived event, an event initiates a request in the traditional definition or trigger to fire a pattern in CEP doesn't apply here. Response time in CEP systems traditionally is measured in this way:

> TRADITIONAL RESPONSE TIME Traditional response time $T_{traditional}$ in CEP is measured from the detection time of the last event T_{last} that triggers the derived event to the time of the derived event $T_{derived}$. It is calculated by:
>
> $$T_{traditional} = T_{derived} - T_{last} \tag{5.1}$$
>
> For example, in Figure 5.1, the traditional response time $T_{traditional}$ is measured as $T_{traditional} = T_{derived} - T_3$.

However, because event pattern detection involves more than one input event, the initialization of a request in pattern detection is not clear. Thus, the start point of the measurement of response time can vary. In another words, response time in such environment can be measured differently.

The response time relative to some other event rather than the last event in matching an event pattern can be useful for system administrators and of interest to users in some event processing applications. It can be the perception of the response time and valuable for system users. The following two applications are examples to illustrate the values of response time to an event which is not the last event in pattern matching.

The first example is a fast flower delivery application from the book [1]: A consortium of flower stores in a large city cooperates with independent van drivers to deliver flowers from stores to their destinations. When a store gets a flower delivery order, the system broadcasts the request for delivery to the drivers close to the store location including the information of required pick-up (typically now) and delivery time. A driver is assigned and the customer is informed of expected delivery time. The driver picks up the flowers and makes the delivery. The person who receives the flowers confirms the delivery time by signing on the driver's device.

The flower delivery process has several processing phases. The assignment phase is one of these phases: the store receives the order, sends a request to the drivers for the upcoming delivery and assigns the delivery to a driver who responds to the request and matches the store's requirements. Figure 5.2 illustrates the pattern detection of the assignment phase. An order, as the input event 1 in the figure, triggers the request for delivery. Meanwhile the order triggers a request for delivery. Responses to the request from drivers in 2 minutes go through a bidding system and get ranked. The information of five highest-ranked drivers is the input event 2 for the store to create an assignment. Following this step, the store makes the decision, creates an assignment and informs the chosen driver. The derived output event is the creation of the new assignment in the system. The derived output event is generated based on the input event 1 and the input event 2. In this case, the response time of the assignment output is traditionally measured from obtaining information of five drivers to the output (i.e., the creation of the assignment). However, the time delay from the moment that the order is received to the creation of the assignment is the response time of the system to a new order from the store's point of view.

The second example is pattern detection of fraudulent use of credit cards that is discussed in [78]: A credit card belonging to a holder resident in California is used in former Czechoslovakia over the internet,

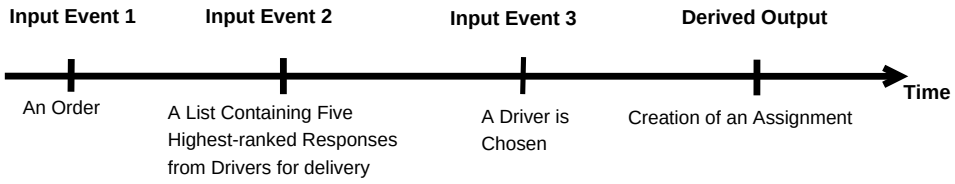


Figure 5.2: The assignment phase in the fast flower delivery application

where the holder has never been to. The card is first used for a transaction of 2 cents over the internet. Following the first transaction, the card is used for increasing transactions of 10 cents, then 2 dollars, 10 dollars and 100 dollars. Each transaction is made after the previous one has cleared processing. As this pattern of fraud is deployed within the card processing systems, an alert is created and the holder is contacted in California. This pattern can be expressed in EPL like the following box and illustrated in the Figure 5.3.

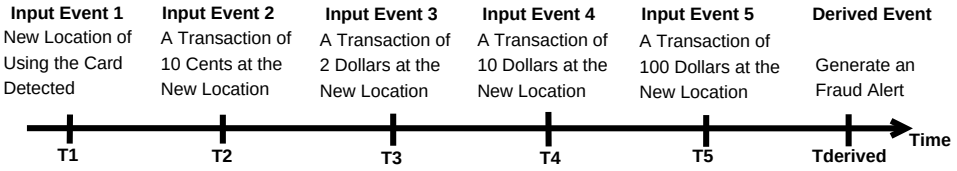


Figure 5.3: The pattern of fraud detection

```
/*
Event pattern of the fraud detection in the second example:
Each transaction event has amount, location and detection time as
attributes
*/
select * from pattern [ every (event1 = (Transaction1
(location != “American”))
    -> (event2 = Transaction1 (amount = event1.amount)
    -> (event3 = Transaction2 (amount > = event2.amount)
    -> (event4 = Transaction3 (amount > = event3.amount))
    -> (event5 = Transaction4 (amount >= event4.amount))
];
```

In this example, the response time that the system takes to react to the last triggering event (a transaction of 100 dollars) is important to prevent further loss. Meanwhile, the earlier the fraud alert is generated after the first fraudulent use of the credit card (the transaction of 2 cents), the less

loss the card holder and the card company make. From the card holder's point of view, the response time that the system takes to react to the first fraudulent use in the defined pattern represents the responsiveness of the service that the system provides.

The measurement of response time to a particular event represents the quality of service that a CEP application delivers from a user's point of view, while the traditional response time cannot. The traditional response time measures how fast a CEP engine processes input events. Comparatively, the response time to a particular event combines the information efficiency measurement which means how fast the system receives the needed information and the CEP engine efficiency measurement which means how fast the CEP engine can process available input events. Therefore, response time to a particular event is not applicable in performance tests that focus on the performance of a CEP engine or comparing performance of different CEP engines.

In the rest of the section, we will introduce the measurement of the response time to the particular event and its application in performance management of CEP systems. We also present an empirical study on applying response time of the particular event to evaluate the performance using CEPBen.

5.1.2 *Response Time of Targeted Event*

A CEP system's responsiveness to a particular event in matching a pattern can represent the system users' perception of response time as it is discussed in the last section. To address this need, we propose the response time of targeted event as an extension to the traditional measurement of a CEP system's responsiveness. The response time of targeted event can be used as a performance metric in CEP systems, in which particular events are critical in deployed event patterns.

> THE RESPONSE TIME OF TARGETED EVENT The response time of targeted event is the response time that a CEP system takes to respond to a particular event in an event pattern. It is measured as the time difference between the detection time

of the targeted event T_{target} and the time of the derived event $T_{derived}$ when the event pattern is matched.

$$T_{response} = T_{derived} - T_{target} \tag{5.2}$$

Response time of targeted event can be applied in Quality of Service (QoS) evaluation in CEP applications.

According to the definition of the response time of targeted event, the response time of targeted event is the same as the traditional response time when the targeted event is the last detected event to match the pattern.

As the Figure 5.4 shows, the response time of targeted event (i.e., Event 1) consists of two parts. One is the time interval from Event 1 to Event 3; The other is the time interval from detection time of Event 3 to the time of the output event. The time interval from event 3 to the time of the output event is equivalent to the traditional response time. While the time interval from Event 1 to Event 3 is the information latency T_{info}, which is defined as follows:

INFORMATION LATENCY Information latency refers to the waiting time from the detection of targeted event T_{target} to the detection of the last required event T_{last} to match a pattern and generate the derived event. It occurs when a CEP engine waits for more required events to be detected after the detection of the targeted event to fully match the event pattern.

According to the definition, the information latency T_{info} can be calculated in this way:

$$T_{info} = T_{last} - T_{target} \tag{5.3}$$

Based on Equation 5.2 and 5.3, T_{target} can be written in Equation 5.4. It shows that response time of targeted event is much affected by the event

input stream of a CEP system. Therefore, response time of targeted event is not applicable in evaluating the performance of a CEP engine.

$$T_{target} = T_{info} + T_{traditional} \tag{5.4}$$

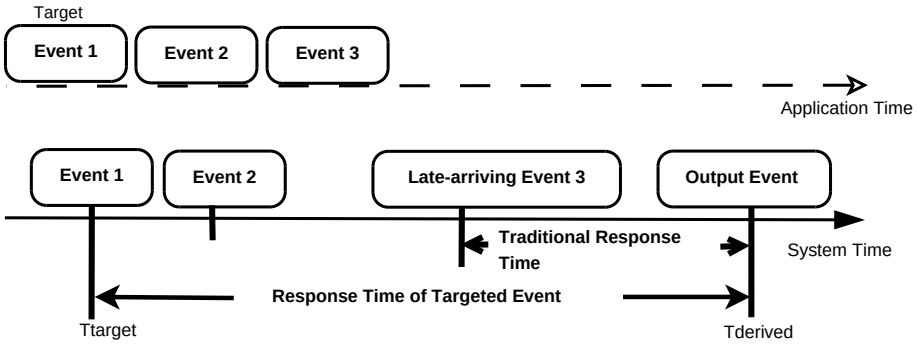


Figure 5.4: Two ways of measuring response time

Figure 5.4 illustrates the two ways of measuring response time: traditional response time and response time of targeted event. A pattern is deployed to detect Event 1 followed by Event 2 and Event 3. In this example, the traditional measurement of response time is the time between the last event (Event 3) to the derived output of the event pattern. If Event 3 is the targeted event, the response time of targeted event is the same measurement as the traditional measurement of response time. If Event 1 is the targeted event, the response time of targeted event is measured as it shown in Figure 5.4.

The targeted event needs to be labelled in the pattern for further processing. The targeted event can be defined differently according to the CEP system specification. Hence, the pattern for Figure 5.4 can be written in the implementation on Esper CEP platform as shown in the three boxes below with different targeted events. The first query statement labels the Event1 as the target. The second query statement labels the Event2 as the target. The third query statement labels the Event3 as the target.

```
/*An event pattern example which defines the targeted event is any
event of Event1 type that has attributeA as 'green' */
select * from pattern [ every TARGET = Event1 (attributeA =
'green')
    -> (Event2 (attributeB = '18'))
    -> (Event3 (attributeC > 3000))];
```

```
/*An event pattern example which defines the targeted event is any
event of Event2 type that has attributeB as '18' */
select * from pattern [ every Event1 (attributeA =
'green')
    -> (TARGET = Event2 (attributeB = '18'))
    -> (Event3 (attributeC > 3 000))];
```

```
/*An event pattern example which defines the targeted event is any
event of Event3 type that has attributeC is larger than 3000 */
select * from pattern [ every Event1 (attributeA =
'green')
    -> (Event2 (attributeB = '18'))
    -> (TARGET = Event3 (attributeC > 3000))];
```

The required information to work response times out can be acquired by the listener for this pattern. The listener for the event pattern continuously receives updated data as soon as the event processing engine generates derived output events when the pattern is matched. The following box contains the listener for the example pattern above. Acquiring information of timestamps of targeted event and the output event is implemented here: Values of attributes in targeted event and derived event can be accessed, so that the timestamps of their detection are used in the calculation of the response time of targeted event. Calculations of response times can be done in the listener like the code below.

```
/* The listener for the event patterns and calculation of response times
*/
public void update(EventBean[] newEvents, EventBean[ ]
oldEvents) {
  if (newEvents = = null) {
    return;
  }
  EventBean detection = newEvents[0];
//System time of the pattern is matched. This will be used to calcu-
late response time.
  long receiveTime = System.nanoTime( );
// Obtain the targeted event when the pattern is matched.
  EventBase event = (EventBase) detection.get("TARGET");
  Object e = event.getSentTime();
  long sentTime = (Long) e;
//Response time of targeted event
  responseTime = (double)(receiveTime -sentTime)/1000;
  (code) //Generate an alert: Event 1(time, ID, attributeA) is detec-
ted.
}
```

In summary, implementing the measurement of response time of targeted event requires modifying pattern queries. One approach to modify the deployed queries is to label the targeted event for events listeners to trace them in the event history. The history of relevant events for matching patterns is kept in stateful event processing system (discussed in Section 1.2.2) and can be used to acquire the details of relevant events. The modification does not change the behaviour of the system, because it does not change the structure of the queries.

5.1.3 *Evaluation on Response Time of Targeted Event*

Following the discussion of the relationship between response time of targeted event and traditional response time, we re-run the experiment that is presented in Section 4.3 to demonstrate the two types of response time measurements. The targeted event is set to be the first event in an event pattern in the experiment tests. Thus, the response time of targeted

event is then measured from the detection time of the first event to the generation time of the derived event when a pattern is matched.

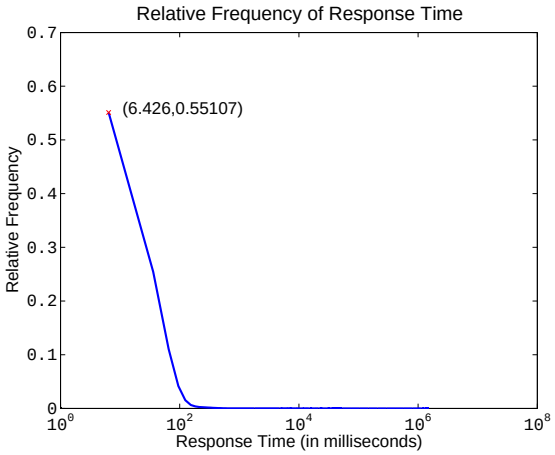
Figure 5.5 and Figure 5.7 show the comparison of relative frequency and cumulative distribution of two types of measurement of response times in Group 1, where 10 event pattern detection queries are registered in the Esper engine. Figure 5.6 and Figure 5.8 show the comparison of relative frequency and cumulative distribution of two types of measurement of response times in Group 2, where 100 event pattern detection queries are deployed in the Esper engine.

All the figures of relative frequency display the shortest response time and the relative frequency of the first bin which contains the shortest response time. The relative frequency distribution for response time of targeted event (shown in the Figure 5.5b and 5.6b) has both a much higher mean and much higher variability than the relative frequency distribution for traditional response time (shown in the Figure 5.5a and 5.6a). Figure 5.7 and Figure 5.8 compare the cumulative distribution of two types of response time measurements.

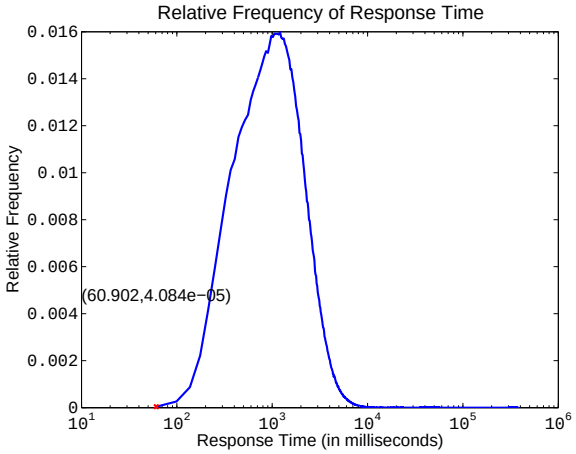
Since the two sets of experiments are run in the same configuration, the significant differences on values of the two types of response time in both groups are caused by the information latency in response time of targeted event. Therefore, it is concluded from the results that the information latency is the dominant factor in the length of response time of targeted event.

To explore the relationship between the traditional response time and workloads, the relationship between the response time of targeted event and workloads and the relationship between the information latency of input events and workloads, the traditional response time, response time of targeted event and the information latency in inputs at various workloads are measured and plotted in the Figure 5.9, 5.11 and 5.10.

The workloads vary from 5,000 events per batch to 25,000 events per batch with an increment of 5,000 events per batch and 0.3 seconds interval between batches. The event pattern is generating a derived event when five events are detected in the defined sequence. The targeted event is the first event that is detected in matching a pattern. 99th percentile of traditional response time, response time of targeted event and the information latency are plotted in the figures to show the changes responding to the workloads.

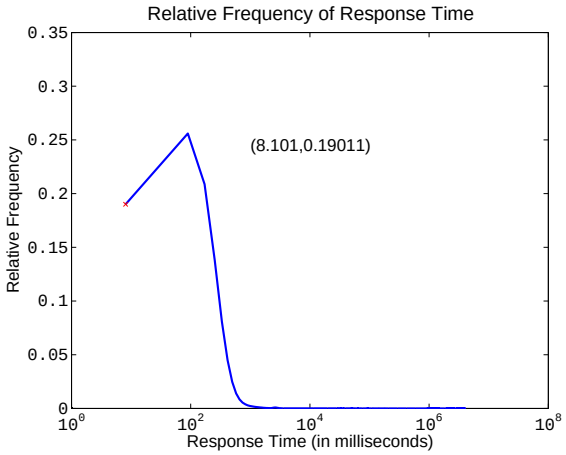


(a) The relative frequency of traditional response times in Group 1

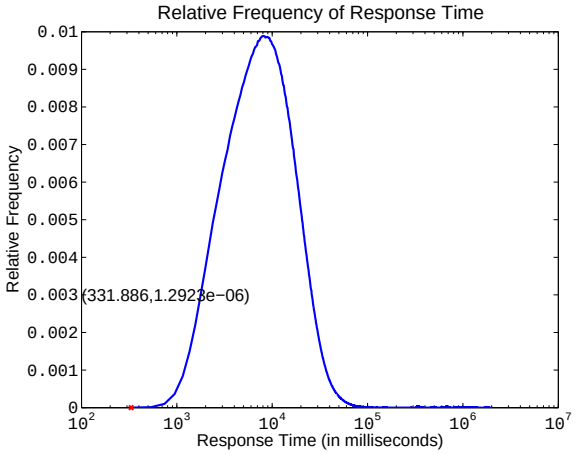


(b) The relative frequency of the response time of targeted event in Group 1

Figure 5.5: The relative frequency of two types of response time in event pattern detection in Group 1



(a) The relative frequency of traditional response times in event pattern detection in Group 2



(b) The relative frequency of response times of targeted event in event pattern detection in Group 2

Figure 5.6: The relative frequency of two types of response time in event pattern detection in Group 2

Percentile is a common statistical criteria in expressing the performance of computer systems. For example, in the Oracle CEP performance study 99.99 percentile is applied to compare latencies [75]. In our experiments, 99th percentile response time is applied to compare the results under different experiments settings.

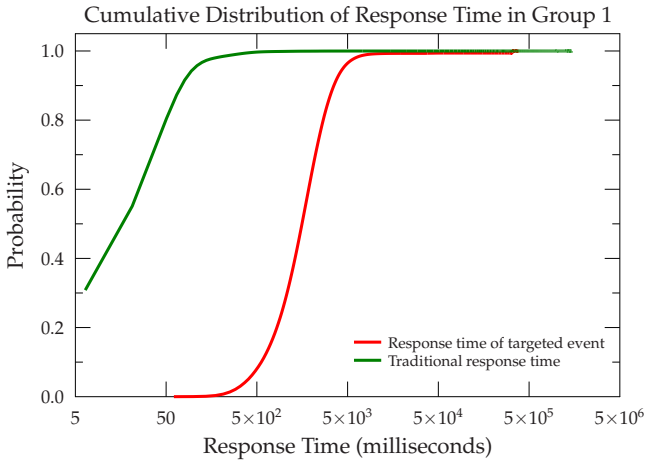


Figure 5.7: The cumulative distribution for comparison of two types of measurement of response time in Group 1

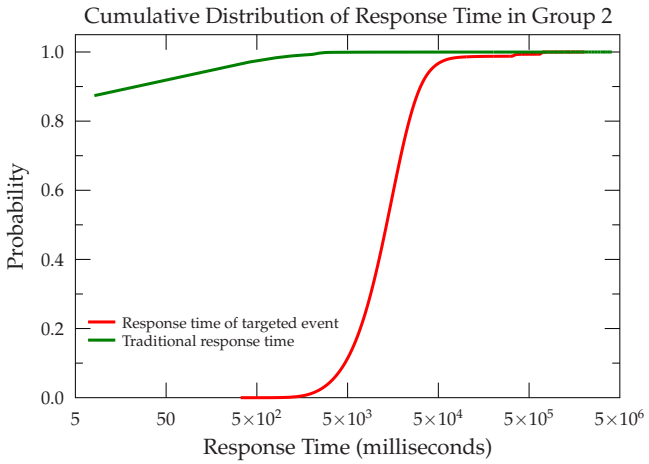


Figure 5.8: The cumulative distribution for comparison of two types of measurement of response time in Group 2

Figure 5.9 describes the 99th percentile of traditional response time with different workloads. The traditional response time T_{system} increases while the workload grows heavier. This complies to typical traditional response time in load testing in computer systems.

Figure 5.10 describes the 99th percentile of the information latency with various workloads. The information latency T_{info} decreases while the workload increases. The reason is that the workloads increases by increasing input events in each batch in the tests. The event types are fixed in the increment, however, the values of the properties of these events are randomly set. By this increment, more relevant events to match the deployed event patterns are detected in the same time period, comparing with less input events in each batch. Therefore, information latency to match an event pattern decreases.

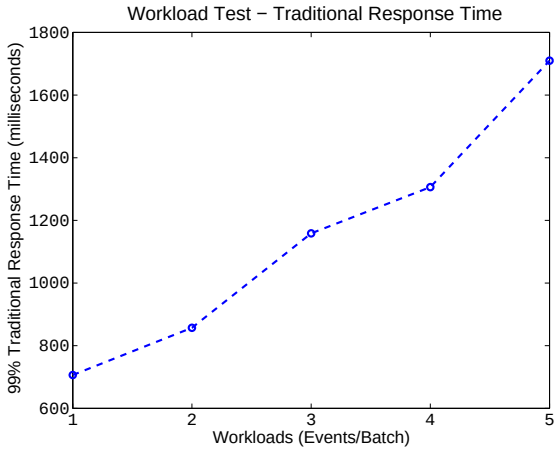


Figure 5.9: The 99th percentile of traditional response time at various workloads

Figure 5.11 shows the 99th percentile of response time of targeted event with different workloads. The response time of targeted event T_{target} decreases while the workload increases. Information latency is a dominant factor to influence the response time of targeted event, because the results from experiments in this section clearly indicate that the length of one information latency measurement is significantly longer that the length of a traditional response time measurement. Even though the traditional response time increases with heavier workloads, the response time of targeted event decreases with heavier workloads because of significant decrease in information latency.

Large information latency indicates that the system takes more resources to maintain and store the event states and history. This is an important factor to consider when allocating memory to event processing

Figure 5.10: The 99 percentile of information latency at various workloads

engines. When the waiting time for more incoming events to a pattern matching is relatively long, more heap needs to be allocated to maintain the event history in order to match queries to guarantee the level of service. When the waiting time for more incoming events is relatively short, more derived events are generated. In this case, more system resources will be consumed on pattern matching and outputting derived events.

The information latency can be obtained by subtracting the traditional response time measurement from response time of targeted event (see Formula 5.3). It can be valuable in an event processing system which takes simultaneous events as input because it is time information related to important events to the system. It not only represents users' perception of response time, but also can be used as a factor to decide whether to deploy workload balancing strategies, when short waiting time is observed continuously and output throughput does not decrease sharply.

Measurement responsiveness and accuracy are the most basic concern for a performance metric [80]. The responsiveness means that a performance metric changes when there is a change in system performance. Accuracy means that the measurement results of a performance metric are close to true values. The response time of targeted event indicates the responsiveness of a CEP system to a particular event in pattern detection. It represents users' perception of the responsiveness of the system service. In this case, the measurement of response time of targeted event

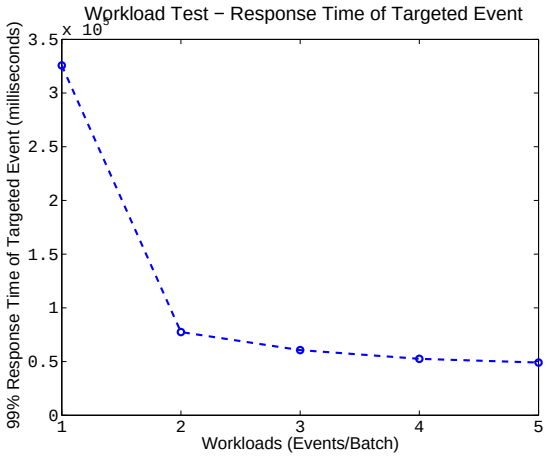


Figure 5.11: The 99 percentile of response time of targeted event at various workloads

is responsive. The changes of response time of targeted event reveal the level of service provided by the CEP system responding to the targeted event. The instrumentation and calculation of response time of targeted event are based on the detection time (i.e., the system time) recorded in the event history. This approach guarantees the accuracy of the measurement of response time of targeted event if the detection time is assigned by the central platform when the events arrive there.

5.1.4 *Summary*

In this section, the response time of targeted event as a performance metric to measure the responsiveness to particular events is proposed and discussed. The need and importance of different measurements of response time in CEP systems is addressed firstly. Then possible measuring points for response time are analysed. At last, the consistency and accuracy of response time of targeted event are evaluated. The two types of response time measurement in the experiments based on the CEP-Ben benchmark are presented and compared. It is concluded that repose time of targeted event is consistent and accurate in indicating the responsiveness to a particular event in pattern detection. The information latency which can be obtained from response time of targeted event and

traditional response time is meaningful in performance management of CEP systems.

5.2 QUERY DEPTH: A PERFORMANCE FACTOR

A query is a language expression that describes data to be retrieved from a database. In a CEP system, wide varieties that the input events can have and the complexity of tasks of event processing suggest the mighty richness and complexity of deployed continuous queries. This richness and complexity of query impact on system performance. In this section, we will explore the performance influence by the complexity of queries.

5.2.1 *Query Depth in Complex Event Processing*

Query processing is recognized as a very important aspect in improving the performance of CEP systems. Various techniques on query processing in CEP to improve the system performance can be found in literature. The development of these techniques shows the importance of query processing for the system performance. For example, a runtime query unsatisfiability (RunSAT) checking technique is developed to detect optimal points for terminating query evaluation in complex event processing [81]. A complex pattern ranking (CPR) framework for specifying top-k pattern queries in complex event processing is proposed to improve the efficiency of systems' pattern matching when plenty of patterns are matched [82]. Instead of focusing the techniques of enhancing query execution, our project focuses on performance effect of queries themselves regardless of the techniques deployed in processing queries.

In complex event processing, events are typically processed in rules by comparing them with other and past events to create complex or abstract events, which are used to signify state changes in relevant entities. Among three main functionalities (filtering, transformation and pattern detection) that is discussed in Chapter 3, transformation produces composite events which are formed based on different number of primitive events. Similarly, event pattern detection can involve various numbers of events. We specify the number of events as query depth. For example, the general example that is described in Figure 5.1, the query depth is 3

because the derived output events are generated based on three events detected in the system.

> QUERY DEPTH Query depth is defined as the number of events that are used to produce a composite event or a derived event in a query statement.

Query depth does not concern about the origins of events. Events in the definition refer to both raw events and derived events in a CEP system. For example, the query of creating an assignment in flower delivery system in Section 5.1.1 has query depth of 3 (5.2). The input event 2 in the pattern is a derived event, while the input event 1 and input event 3 in the figure are raw events.

Sometimes more than one attributes of a single event can be used in a query statement. For example, the statement that detects the fraudulent use of a credit card in Section 5.1.1 (shown in the box below) is written in a way that it seems involving 5 events: event1, event2, event 3, event4 and event5. In fact, only four transaction events are used to match this pattern. "event1" and "event 2" in this query are two different attributes of one transaction event. In this case, the query depth refers to the number of events, not the number of attributes. Hence, this query statement has query depth of 4. The events in this statement are all raw events.

```
/*
Event pattern of the fraud detection in the second example:
Each transaction event has amount, location and detection time as
attributes
*/
select * from pattern [ every (event1 = (Transaction1
(location != "American"))
    -> (event2 = Transaction1 (amount = event1.amount)
    -> (event3 = Transaction2 (amount > = event2.amount)
    -> (event4 = Transaction3 (amount > = event3.amount))
    -> (event5 = Transaction4 (amount >= event4.amount))
];
```

Query depth is important because it refers to the event history that a CEP engine needs to maintain for processing query statements that involve more that one primitive event. Maintaining the history is resource-consuming and affects the performance of a CEP system. In this section,

we will investigate the query depth as a performance factor in performance management of CEP system.

5.2.2 *Performance Analysis on the Factor of Query Depth*

In the Section 4.3, two groups of experiments are designed and presented. In the experiments according to the definition of query depth, the query statements of filtering has query depth as 1, the query statements of transformation has query depth as 2 and the query statements of pattern detection has query depth as 5. Figure 4.5 and Figure 4.9 show the response time of the three types of queries. Filtering with the least query depth performs the best in the two groups of experiments. In Figure 4.10 and Figure 4.11, the event pattern detection with query depth of 5 has higher input throughput than the transformation with query depth of 2. The reason might be the output throughput is much heavier in transformation than in event pattern detection as shown in Table 4.2 and Table 4.3. Higher output throughput means queries are matched and fired for more times and more derived events are generated in the measured period. This indicates heavier processing load for the CEP engine. Therefore, the performance impact from higher query depth in pattern detection is offset by the performance impact from the heavier processing load in the transformation.

The results of Group 1 and Group 2 demonstrate the differences that the different type of functional queries with different query depths make on performance. However, these two group of tests are not enough to exclude the possible reason that the event processing engine's nature on different functionalities. To clarify the influence of query depth, another group of test is designed: Group 3. In the Group 3, a single functionality — event pattern detection, with two query depth settings which are 3 and 5, is tested. The workload is the same as the Group 1 and Group 2: It consists of 500 event batches with 20,000 events in each batch; The average of interval times is set to 0.3 seconds. In addition, 100 queries are deployed in this group of experiment.

Figure 5.12 and 5.13 describe the cumulative distribution of traditional response time and response time of targeted event in Group 3. The targeted event is the first detected event of each deployed query statement. The relative frequency distribution is divided into two parts in

order to show the curve clearly. The part of the distribution that goes beyond the visible part of the graph is not displayed here. The CEP system performs better with queries that have query depth as 3 than queries that have query depth as 5 in evaluation of traditional response times. Although the CEP system performs has better performance with queries that have query depth as 3 than queries that have query depth as 5 in the evaluation of response times of targeted event with the first event of each pattern as the targeted event, choosing targeted events in different queries according to real-world application scenarios makes a lot of differences in the value of response time of targeted event.

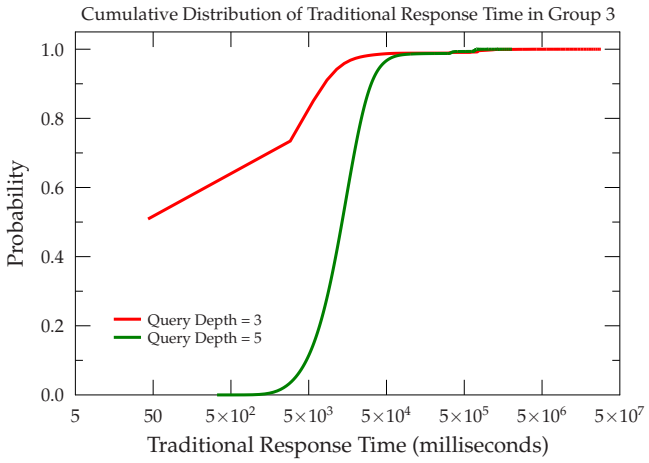


Figure 5.12: The cumulative distribution of traditional response time in the Group 3

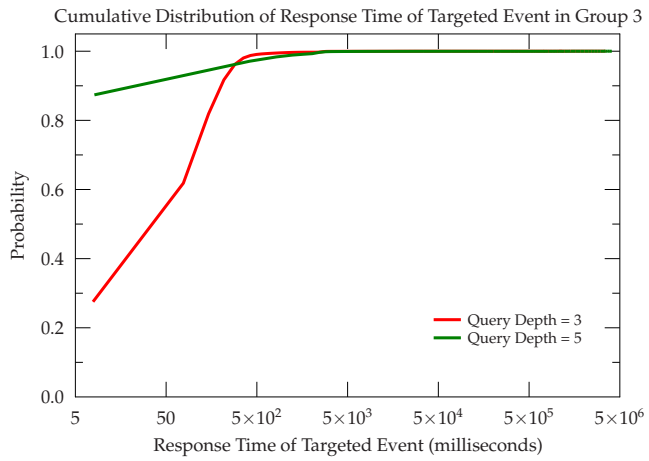


Figure 5.13: The cumulative distribution of response time of targeted event in the Group 3

Figure 5.14 provides the input throughput in the Group 3. The measurement of input throughput is taken in each event batch. Therefore, there are 500 measures in each run of an experiment. The X axis of Figure 5.14 is the measures taken in 500 event batches in a time series. The system with query depth of 3 has much higher input throughput than the system with query depth of 5. A significant drop in the input throughput is found in both input throughput lines in the graph.

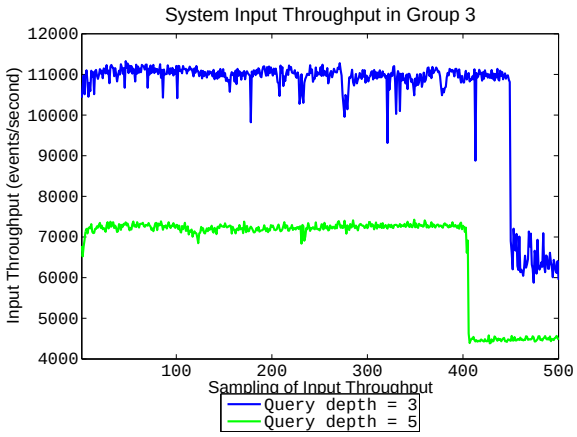


Figure 5.14: The Input throughput in Test group 3

To investigate the reasons that cause such phenomenon, the CPU usage, garbage collection and heap size are monitored. Figure 5.15 and 5.16 are screen shots that are caught during two experiments in Group 3. The X axis is the time when the experiments are run in these screen shots. in the As it is shown in the graphs, the CPU usage, garbage collection and heap size had steady increase in the later period of the experiment. At the final stage, the used heap is very close to the maximum of the heap size. This stress on the heap results in the drop in input throughput in Figure 5.14.

In contrast, Figure 5.17 shows a screen shot of typical system resources usage of a filtering experiment. The deployed queries are 100 filtering statements in this figure. The CPU usage and heap usage remains in a steady level through the experiment. The resource consumption is very different from the increasing resource consumption in Group 3.

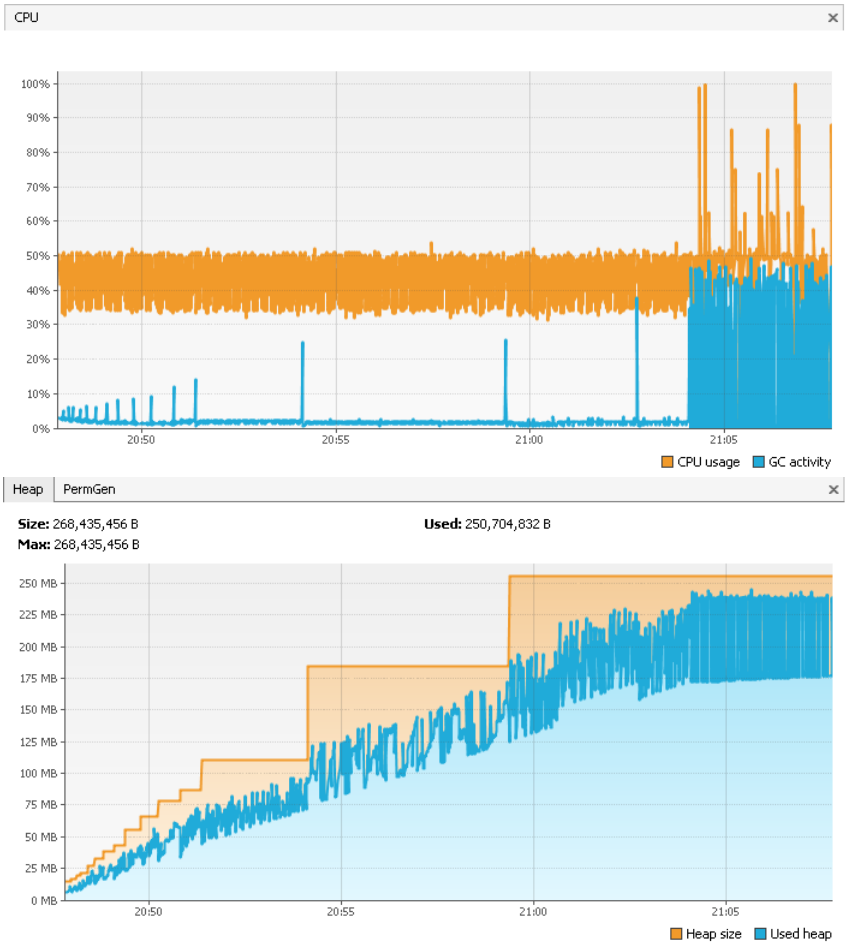


Figure 5.15: A screen shot of VisualVM in the running system with queries of query depth 3

Hence, it is concluded by the comparison above that the increase in heap size in Group 3 (shown in Figure 5.15 and 5.16) can be caused by the accumulation of event history in the CEP engine. Because the queries that are deployed in the experiments are not specified with window time, the states of relative events in early period of the experiment can be maintained in the CEP engine until the later period when more input events are detected for matching the related event patterns. When the heap size reaches the maximum, more garbage collection is performed to release memory for processing. This is observed in both Figure 5.15 and 5.16.

Table 5.1 shows the output throughput in the Group 3. The system generates larger amount of output events and higher output throughput in the experiments with queries that have query depth 3 than the expe-

riments with queries that have query depth 5. Larger amount of output events and higher output throughput indicate that more events match the deployed patterns and heavier processing load in the CEP engine. Figure 5.15 and 5.16 show the heap is stressed more in the experiment with query depth as 3 than the experiment with query depth as 5. This observation confirms that the performance influence by larger amount of output events and higher output throughput.

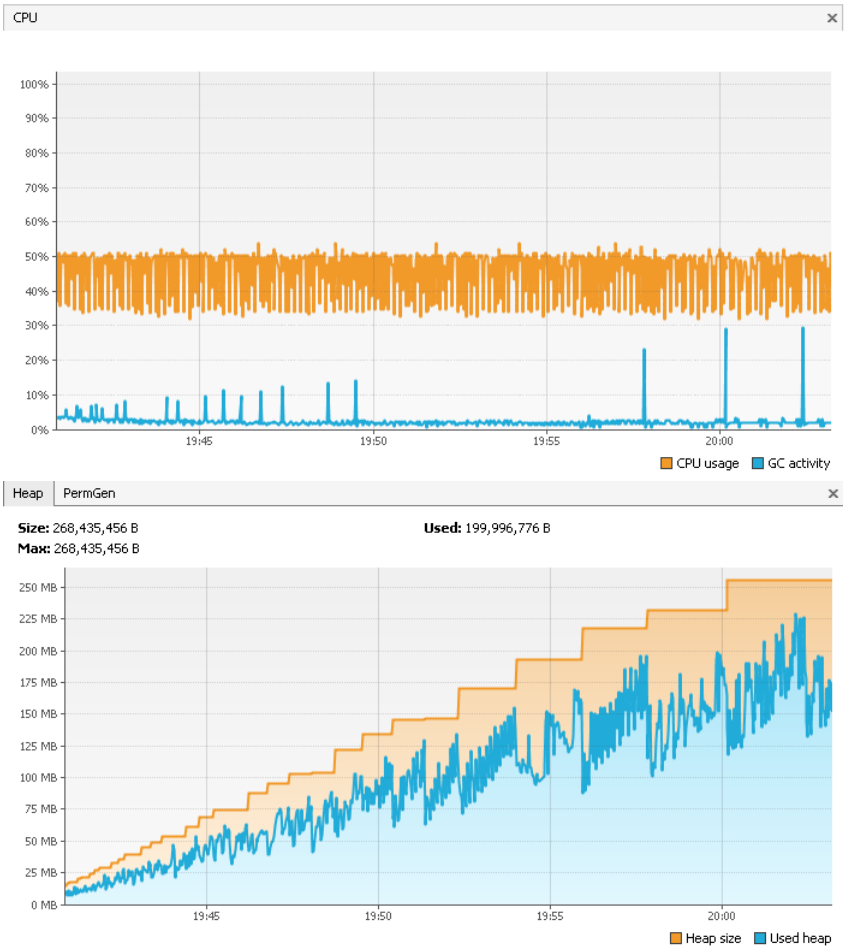


Figure 5.16: A screen shot of VisualVM in the running system with queries of query depth 5

In a summary, query depth significantly influences the performance of CEP systems. Comparing with the tested system with queries that have more query depth, the system with queries that have less query depth has better response time and higher input throughput. It is noticed that the system with queries that have less query depth deals with heavier

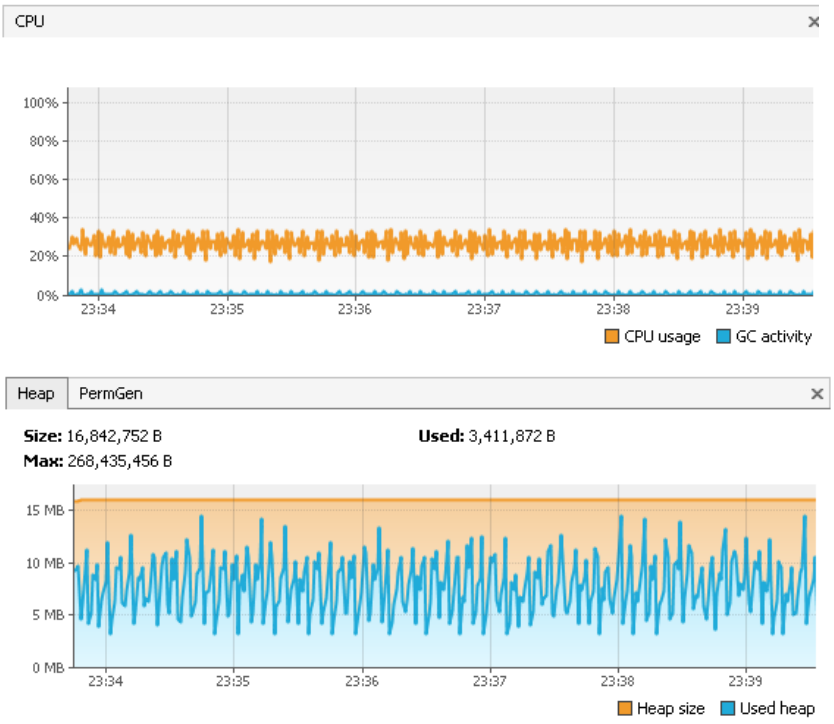


Figure 5.17: Typical CPU usage, garbage collection activity and heap usage in a filtering experiment

output load and generates higher output throughput to process the same amount of input events in the experiments above.

Table 5.1: The output throughput in Group 3

Query Depth	Output of Group 3	
	Throughput (events/sec)	Load (events)
3	13639.01	18,102,554
5	6866.44	10,549,305

5.2.3 *Summary*

In this section, query depth as a factor that influence the performance of CEP systems is investigated. A group of performance test is designed to explore the impact. Detailed results obtained from the experiments were described and analysed. It is concluded query depth has impact on the

performance of CEP systems. Systems with less query depth of queries perform better than systems with more query depth of queries.

5.3 QUERY LOAD

In a CEP system, the load of input events is an important factor that influences the system performance. Moreover, queries in CEP systems are continuous queries which are issued once and then logically run continuously over the data. The queries deployed in a CEP system forms a load in the system. In this section, we will explore the impact on the system performance that is made by the load of queries that are deployed in a CEP system. The experimental work done on Esper using CEPBen is presented to demonstrate the impact of varying number of queries on the system performance.

5.3.1 *A Capacity Indicator: Maximum Query Load*

The queries in complex event processing engines are commonly known as continuous queries. Multiple continuous queries are usually registered simultaneously in a CEP system. As it is observed that processing continuous queries over data streams involves fundamental trade-offs among efficiency, accuracy and storage, Dobra A. et. al states that it is an open problem to understand the implications of these trade-offs in a real system processing continuous queries [83].

Furthermore, from performance point of view, executing such queries is expensive. When the system resources are allocated to the execution largely, the level of Quality of Service (QoS) (e.g., response time, input throughput and output throughput) that is provided by this system will degrade. Therefore, knowing the number of queries that a CEP system can support and at the same time providing a satisfying level of QoS is important.

The Linear Road Benchmark that is reviewed in Section 2.3 used "supported query load" as a metric to address the challenge of continuous queries [63]. The authors argue that the typical database benchmark metric — completion time — is not appropriate for continuous queries because such queries never complete. Supported query load is defined as

the input that a stream system can process while meeting specified response times and correctness constraints.

Different from the Linear Road Benchmark, the number of queries that a CEP system can support is also interesting because it reflects the capabilities of the CEP system on processing events. Deploying queries is a dynamic process in CEP systems. For example, in Esper a query statement can have three states: Started: when the query statement is actively evaluating input events according to the queries expression; Stopped: when the query statement is inactive; Destroyed: when query statement resource is relinquished. Therefore, we define query load as the following:

> QUERY LOAD Query load is the number of query statements that are active at the runtime while the CEP engine is meeting specified performance requirements. Query load consists of three components reflecting to the three main functionalities a CEP engine operates: a number of filtering queries, a number of transformation queries and a number of pattern detection queries.

The more active queries a CEP system can support, the better capacity the CEP system has. Therefore, we propose maximum query load as a capacity indicator for event processing systems.

> MAXIMUM QUERY LOAD Maximum query load refers to the maximum number of active queries that a CEP engine can support while providing a satisfied service. The maximum query load can be expressed including two figures: total number of active queries and the ratio of numbers of three types of functionality queries (i.e., number of filtering queries: number of transformation queries: number of pattern detection queries). Maximum query load of a CEP engine is affected by the workload and the query depths of the deployed queries in the tested CEP system.

Query load in a CEP system is important for both users and developers in guaranteeing a promising responsive service. From user's point of view, having knowledge of the query load that can be supported by candidate CEP systems is helpful to estimate and choose the right CEP system to run their application when making decisions and to avoid

unnecessary service failure in running systems. From developers' point of view, having knowledge of the query load supported by their CEP systems is beneficial to detect bottlenecks and improve the performance to meet users' requirements on performance.

In the next section, we will introduce the implementation of query load. Experiment results from the CEPBen benchmark that is implemented on Esper are discussed on the impact of various query loads on the system performance.

5.3.2 *Implementation and Experiments of Query Load*

The number of active queries that are deployed in a CEP system can be tracked either via built-in interfaces at runtime if the CEP engine interface has the method to return the value of the number of active queries, or implementing a class to count the state changes when new query statements are deployed and the states of query statements are changed. In our experiment, the number of deployed query statements stays the same from the beginning to the end of the experiment.

To demonstrate the performance impact by query load, the results that are obtained from the experiments described in Section 4.3 are presented. The input workloads in this set of experiments are set as the following: 20,000 events in each event batch, 500 event batches in total, and 0.3 seconds as the interval between two events batches. Two query loads are implemented in the two groups of tests: 10 queries and 100 queries. Cumulative distribution of traditional response time with three types of functionalities and response time of targeted event with pattern detection are plotted. The targeted event is set to be the first event detected to match a pattern.

Figure 5.18 compares the response times acquired in the CEP system run with 10 filtering queries and 100 filtering queries respectively. Figure 5.19 compares the response times in the CEP system with 10 transformation queries and 100 transformation queries respectively. Figure 5.20 and 5.21 compare the traditional response times and response times of targeted event in the CEP system with 10 pattern detection queries and 100 pattern detection queries respectively. These figures reveal the impact of different query load on response time: Higher query load leads to longer response time.

The response time of targeted event represents the users' perception of response time. Therefore, the response time of targeted event is particularly important in deciding whether the agreed level of service is provided in a CEP system.

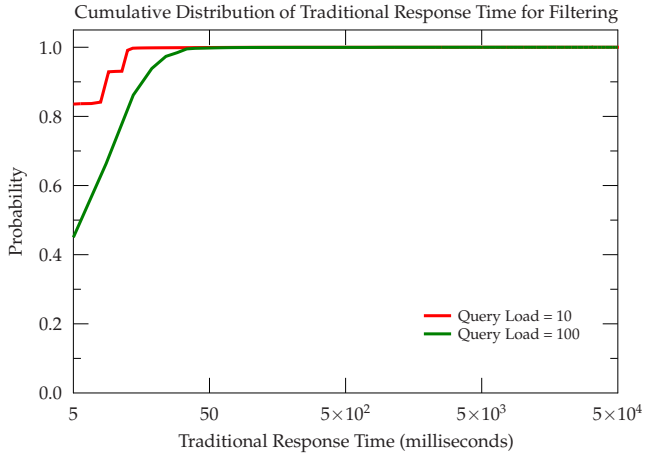


Figure 5.18: The cumulative distribution of response time for filtering with different query loads

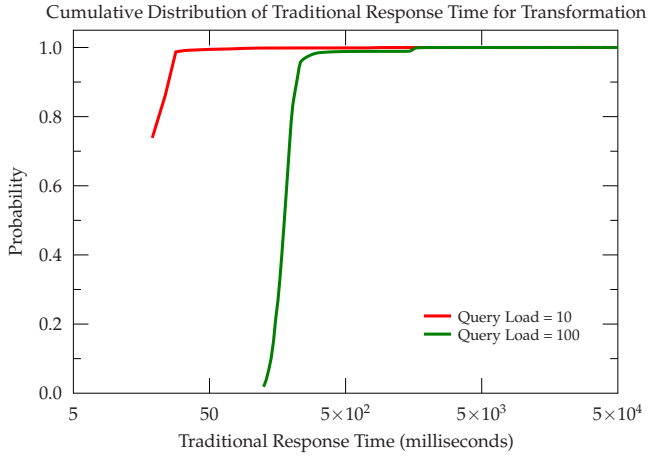


Figure 5.19: The cumulative distribution of response time for transformation with different query loads

Trade-off study between 99th percentile response time and query load and trade-off study between average input throughput and query load are studied and presented in Figure 5.22 and 5.23, to compare the system performance under various query loads. 40 queries and 70 queries as two query loads are added in the trade-off study to create a steady increasing query load setting: 10—40—70—100 queries. With each query

load setting, filtering, transformation and pattern detection queries are deployed respectively and displayed in one graph. Note that the trade-off between average output throughput and query load is not displayed here, because the number of output events and the output throughput heavily depend on the context of deployed queries in the system against input events.

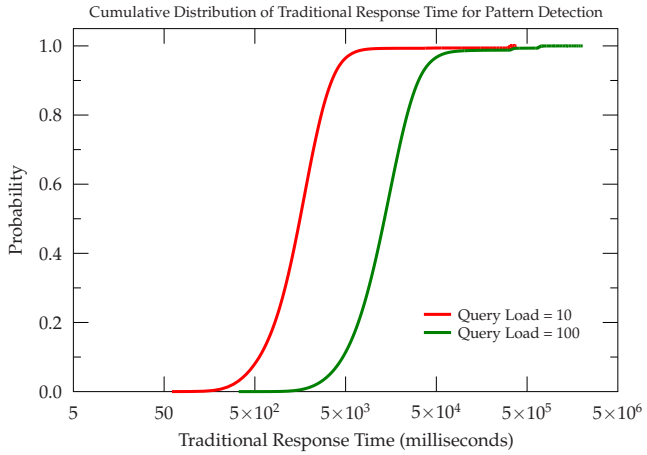


Figure 5.20: The cumulative distribution of traditional response time for pattern detection with different query loads

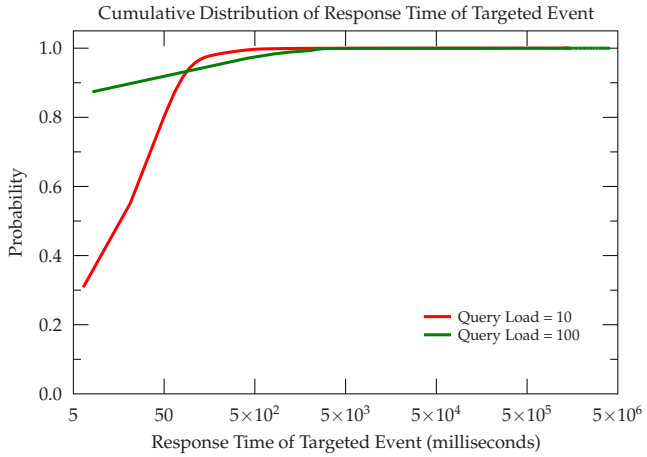


Figure 5.21: The cumulative distribution of response time of targeted event for pattern detection with different query loads

Figure 5.22 demonstrates the trade-off between query load and 99th percentile traditional response time. 99th percentile traditional response time refers to 99th percentile of the shortest response times among all the response times that are produced in one set of experiments. As it

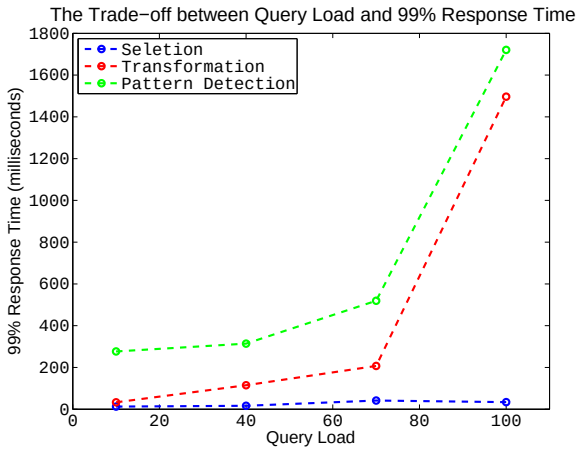


Figure 5.22: The trade-off between 99% response time and query load

is shown in the graph, the traditional response time increases while the query load increases.

Figure 5.23 shows the trade-off between query load and average input throughput. Input throughput is measured once in each event batch, therefore, the average input throughput of a run of an experiment can be calculated based on these data. Several runs of experiments are carried out. The average input throughput is calculated and displayed in the graph based on average input throughput of each run of experiments. It is concluded from Figure 5.23 that the average input throughput is in inverse relation to the query load.

The experiment results show that query load has significant impact on performance. The more the query load is, the higher response time that the system performs, the lower average input throughput that the system processes. Therefore, maximum query load can be used to indicate the capacity of a CEP system under the constraint of QoS. The trade-off studies are helpful for developers and users to judge the performance level at runtime in the performance management. As an outcome of this query load study, it is recommended to include trade-off study of query load in performance reports of CEP systems.

To compare the capacity of different CEP systems using maximum query load, the total number of active queries and the ratio of number

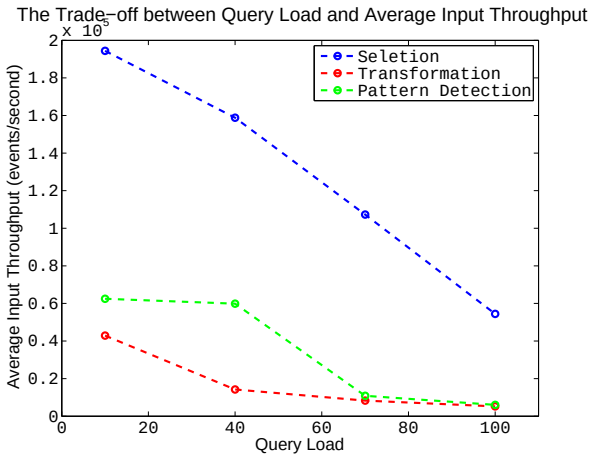


Figure 5.23: The trade-off between average input throughput and query load

of the three types of functionality queries are both considered. According to the results that are obtained from experiments, the capability of a CEP system descends in supporting different types of queries in the following sequence: filtering, transformation, and pattern detection. Therefore, the more pattern detection queries in the maximum query load that a CEP system can support while meeting the agreed level of service, the more capable the CEP system is. If the numbers of pattern detection queries are the same, the more transformation queries in the maximum query load that a CEP system can support, the higher capability the CEP system has. If numbers of pattern detection queries and numbers of transformation queries are the same in two CEP systems, then the numbers of filtering queries in the maximum query loads of these two systems are compared.

5.3.3 *Summary*

In this section, the definition and the significance of query load and maximum query load in CEP systems are presented. Then the performance impact by query load is studied and demonstrated using the CEPBen benchmark on Esper. Trade-off studies between query load and 99 percentile response time, and the trade-off between query load and

input throughput were presented to show the performance influence by query load. It is concluded that maximum query load can be used as a capacity indicator for CEP systems. Trade-off studies regarding to query load and other performance metrics are recommended in measuring performance of CEP systems. At the end, applying the maximum query load to compare the capability of different CEP systems is discussed.

5.4 LIVE OBJECTS IN HEAP

Memory management is critical in performance management of CEP systems. It is known that memory management plays an important role in meeting the agreed level of service with limited resources.

In this section, we will explore performance of memory management in CEP and propose the number of live objects in heap as a performance metric to indicate the usage of the heap.

5.4.1 *Garbage Collection*

Memory management is the process of allocating new objects and removing unused objects to make space for those new object allocations [84]. Garbage collection (GC) is a memory management technique for automatic reclamation of allocated storage for programs. Many computer languages require garbage collection as part of the language specification, e.g., Java, C#, while some languages use manual memory management, but have garbage collected implementations available (e.g., C, C++) [85].

Garbage collection relieves programmers from memory management and reduces certain categories of bugs. However, garbage collection has certain disadvantages. Garbage collection consumes computing resources in deciding which memory to release. Furthermore, it can cause unpredictable stalls in real-time environment, transaction processing or interactive programs [86]. This disadvantage is very critical for complex event processing when timely responses are required.

To study the garbage collection's impact on CEP, a group of performance tests on CEPBen is set up. The garbage collection activity in the tests is monitored. The garbage collection in the experiments automatically occurs because CEPBen is implemented in Java and the garbage

collection is built in Java Virtual Machine (JVM). In this test, the input workload consists of 500 event batches with 20,000 events in each batch. Five pattern detection queries are deployed. Each query has query depth as 3. VisualVM (introduced in Section 4.3.3.4) is used to monitor the garbage collection activity. Traditional response time and input throughput are measured. As the input throughput is measured once in each batch, 500 input throughput data are obtained in each run of the experiment.

An experiment with the settings described above is run. The performance data is plotted in Figure 5.24, 5.25, and 5.26 . Figure 5.24 is a snapshot of VisualVM that describes garbage collection activity and CPU activity in one run of the experiment with the described experimental settings. It shows that the garbage collection activity is active all the time including several peaks at the beginning of the experiment (warm-up period), 12:41:40, 12:42:15 and 12: 43:10 roughly estimated. The garbage collector in the machine where the CEP system runs is generational garbage collector [87].

Figure 5.25 depicts 1 percentile worst traditional response times in time sequence in this experiment. It is found that worst response times occur at different times in the experiment. Moreover, when the garbage activity peaks happen, long response times are detected at around the peaks' time. These are marked out in the Figure 5.25 with underlined time periods.

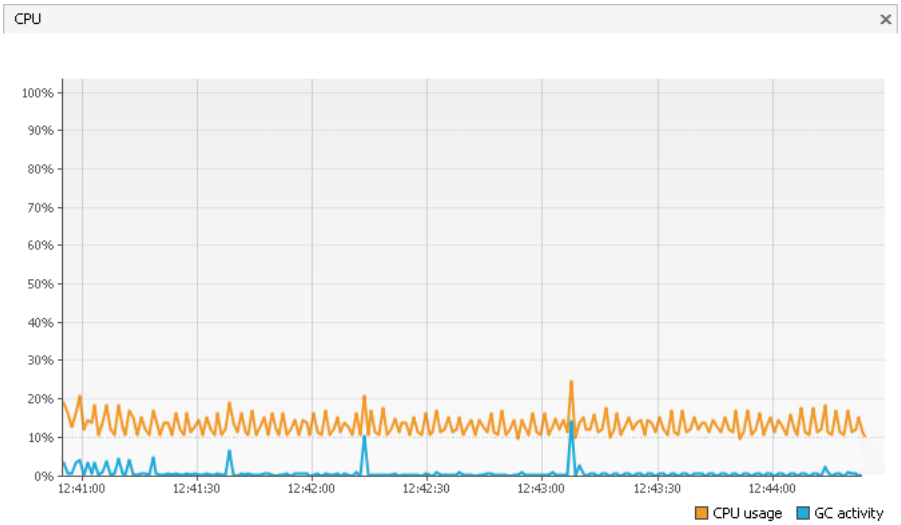


Figure 5.24: Garbage collection activity (an example)

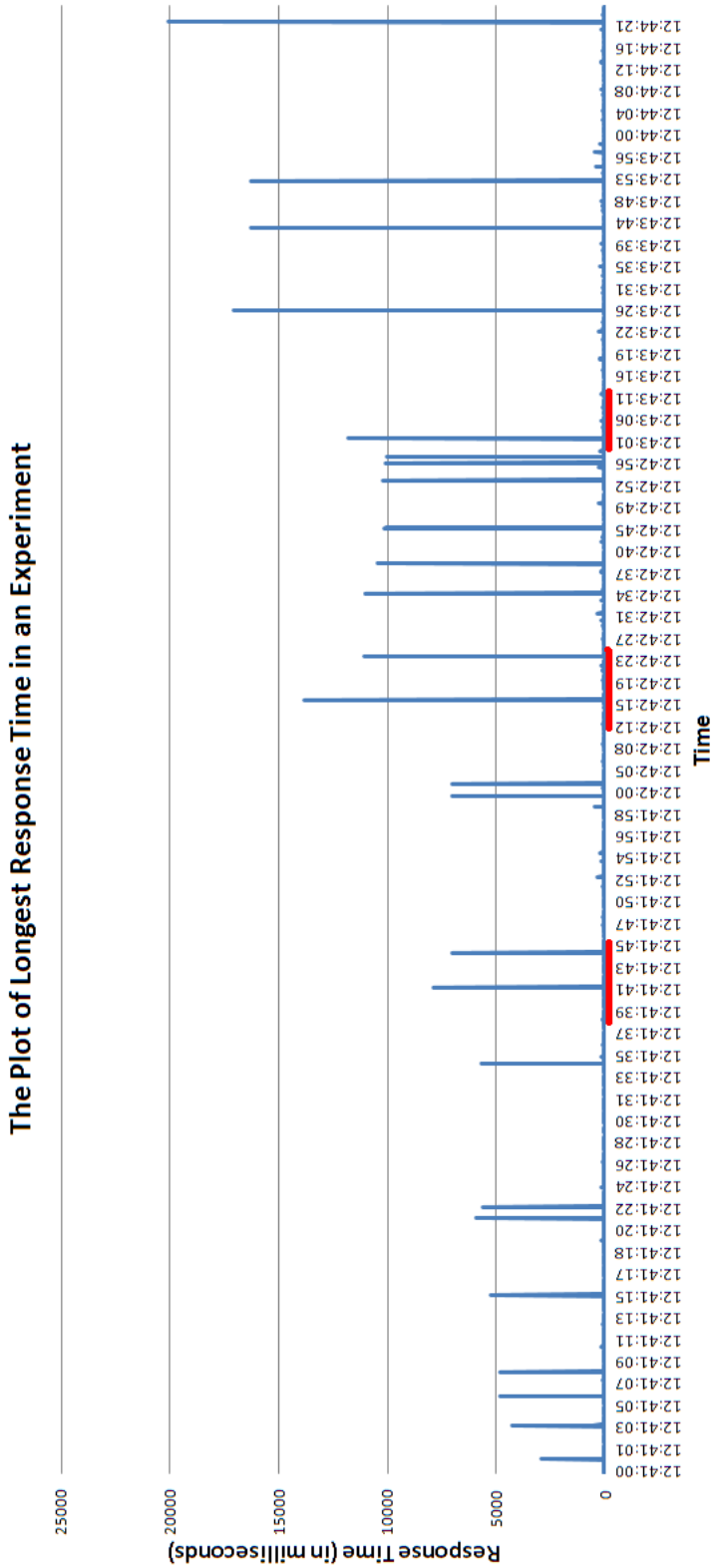


Figure 5.25: One percentile of worst traditional response time

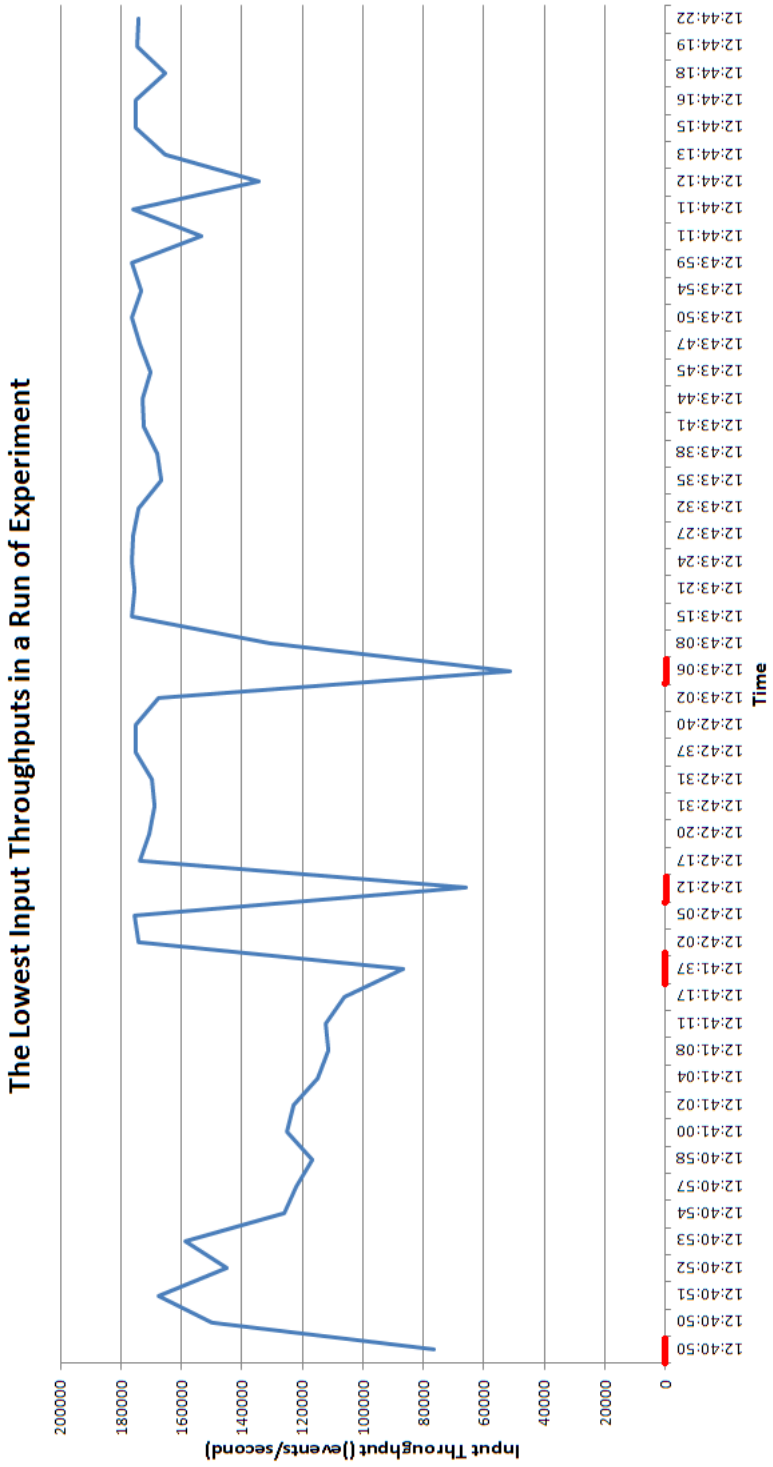


Figure 5.26: 10 percentile of worst input throughput

Figure 5.26 records 10 percentile worst input throughput in time sequence. Because one measurement of input throughput is taken measured in each event batch and there are 500 event batches in total, 10 percentile worst input throughputs contain 50 input throughput data. Comparing with the garbage collection activity, significantly drops of input throughput measurements occur at 12:41:37, 12:42:12 and 12:43:06, corresponding to the garbage collection activity peaks.

Garbage collection activity can have a significant impact on performance of CEP systems. The correspondence between the time of garbage collection activity peaks and the time when the lowest input throughput and highest response times occurred at runtime demonstrate the influence of garbage collection on the CEP system performance.

Therefore, for better and precise measurement of the performance in CEP systems, we will explore a new way of measuring the memory usage considering the influences from memory management techniques in such systems in the rest of this section.

5.4.2 *Live objects as a Performance Metric*

In the previous section, it is demonstrated that garbage collection plays an important role in the performance management of CEP systems. Extreme performance in real-time processing can cause unacceptable services at critical times.

There are many different dynamic memory management schemes and garbage collection algorithms [85, 88]. Moreover, different implementations of memory management systems apply different memory management schemes and garbage collection algorithms. No matter what dynamic memory management scheme and garbage collection algorithm are adopted in memory management, there are common issues in the heap.

Java objects reside in the heap. The heap is created when the JVM starts up. The usage of the heap may increase and decrease in size when applications run. The heap consists of reachable objects and unreachable objects. Reachable objects are live objects and actively used by the system. Unreachable objects in the heap are ready to be collected. The heap usage reflects the current usage of the heap. However, considering the frequently occurred garbage collection in the memory, the heap that is occupied by unreachable objects can be released as soon as garbage col-

lection happens. Consequently, a drop in used heap occurs after the garbage collection. For example, a snapshot of the heap usage taken in an experiment is displayed in Figure 5.27. The maximum heap size is 256 megabytes (MB). The heap usage reaches 70% (about 200MB) at about the time 21:00. As time passes, the used heap is growing larger until it reaches the maximum heap size. Even though the used heap fluctuates at the level that is close to the maximum heap size, the experiment is running well. This is because the garbage collection can release memory for the system. At this stage when the heap is under excessive pressure, the number of live objects in the heap can be used to reveal the capability of the system to guarantee the service.

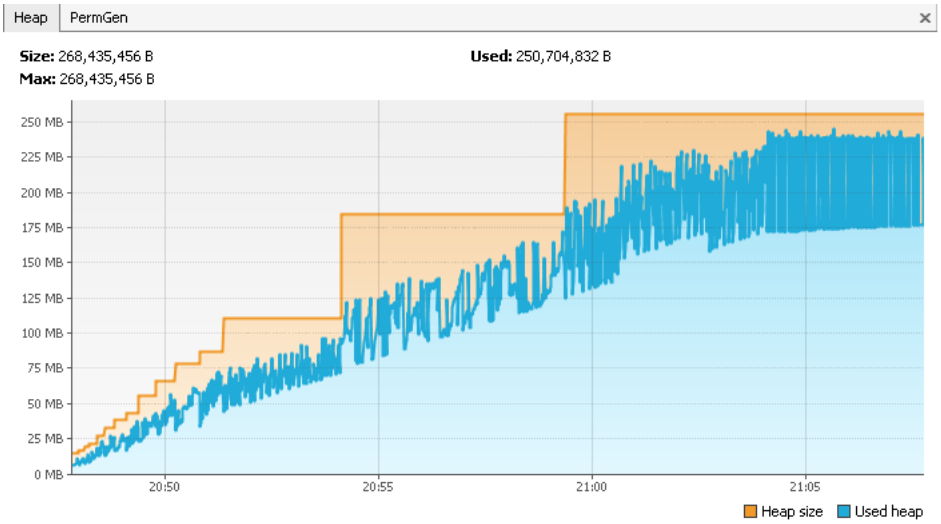


Figure 5.27: The heap under excessive use

On the other hand, the number of unreachable objects can indicate the imminent initiation of garbage collection activity, because they trigger garbage collection activity. The number of unreachable objects in the heap can help predicting garbage collection activity. Although the performance impact cannot be predicted during the garbage collection, predicting high garbage collection activity offers opportunities to the system to get prepared for possible unacceptable system performance that results from garbage collection.

CEP systems are featured of real-time processing. The memory in such systems is highly demanded in processing events and storing event history for matching patterns. High input throughput and output throughput, large query load and the query depths of these queries deeply affect

the memory consumption. Therefore, to capture the characteristics of the heap usage in CEP systems, we propose the ratio between number of live objects and the total number of objects in the heap (i.e., live objects and unreachable objects) as a performance metric. The ratio is calculated in Formula 5.5.

$$\gamma = \frac{N_{live}}{N_{live} + N_{unreachable}} \tag{5.5}$$

Where γ is the ratio of the live objects in the heap, N_{live} is number of live objects, $N_{unreachable}$ is the number of unreachable objects in the heap. According to the definition of the ratio of the live objects, the more the number of live objects is, the higher γ is, in fixed size of heap.

In the next section, we will present the implementation and analysis of applying the number of live objects in the heap as a performance metric.

5.4.3 *Implementation and Performance Analysis*

The number of live objects can be obtained by heap dumps. A heap dump is a snapshot of the main memory. It can be created via JVM functions or by using special tools that utilize the JVM tooling interface. The heap dump itself contains rich information about the memory, including the information of number of live objects and unreachable objects. The rich information generally is used in identification of memory leaks and memory-eaters in a program.

However, generating a heap dump requires memory itself. In addition, a heap dump will suspend the JVM [89]. Therefore, heap dump cannot be done under very heavy load or in real-time. An experiment based on CEPBen with 100 queries deployed is run. All the queries have query depth as 5. The workload of the experiment is set the same as the experiments in Chapter 3: 20,000 events in each batch and 500 batches in all. The heap usage, CPU usage and garbage collection activity is monitored with VisualVM. The heap dump is acquired and analysed by Memory Analyzer (MAT) [1]. To avoid problems of generating heap dump when

1 The Eclipse Memory analyser is a fast and feature-rich Java heap analyser. MAT

the heap is excessively used, the maximum heap size is set to 512 MB which is doubled than the experiment presented in Figure 5.27.

Figure 5.28 and 5.29 depict the CPU usage, garbage collection and heap usage in the experiment. Figure 5.28 reveals the moments that three heap dumps are acquired. The CPU usage has three spikes at different times. The used heap has steady growth in Figure 5.29. Figure 5.30 shows that the number of live objects increases steadily, corresponding to the used heap size in Figure 5.29. The number of unreachable objects indicates the memory that can be released after garbage collection. Hence, the ratio of live objects in the heap can be plotted based on these data (shown in Figure 5.31). The ratio increases while the used heap increases. The figure of the ratio can indicate the state of the memory usage. The bigger the ratio is, the less heap memory is available for use.

The memory usage becomes a constraint for achieving promised service (e.g. sound response time, high input throughput and high maximum query load) when the heap is excessively used. The ratio of live objects in the heap can be used as a performance metric to describe the used heap in CEP systems. It is consistent as it increases while the used heap increases. Moreover, it is more precise than the used heap in revealing the available memory to use in CEP systems.

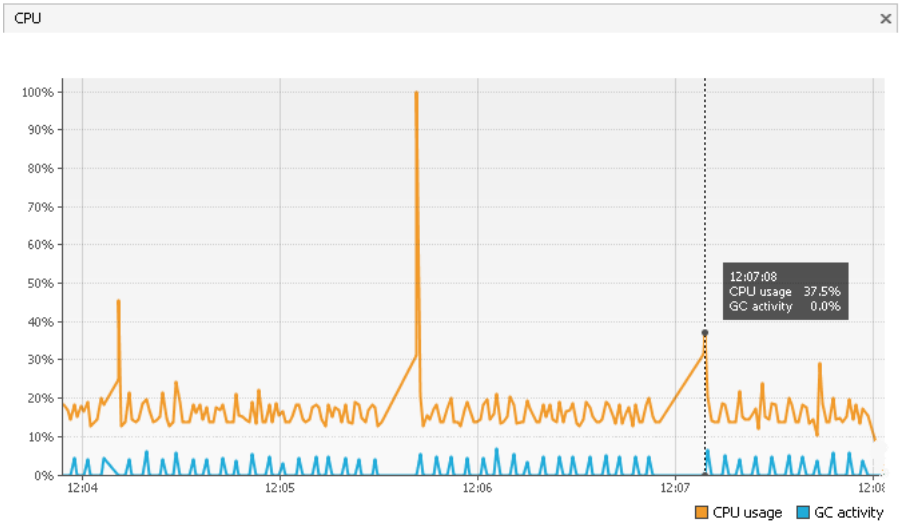


Figure 5.28: The CPU usage and garbage collection activity in the experiment

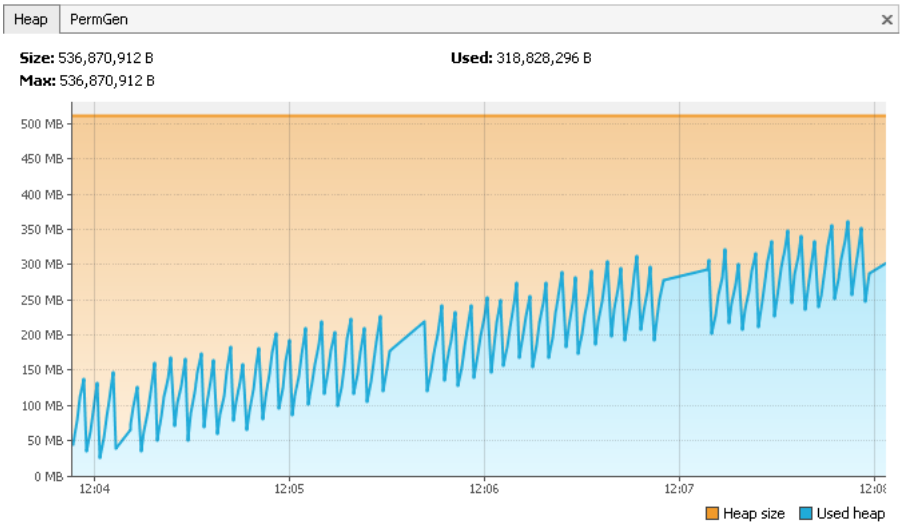


Figure 5.29: The heap usage in the experiment

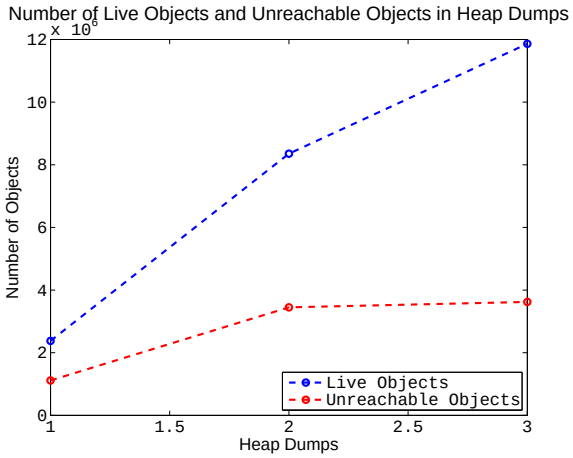


Figure 5.30: Number of live objects and unreachable objects in three heap dumps in the experiment

Furthermore, the ratio of live objects in the heap is especially useful because it is associated with query depth. The ratio of live objects implicates the memory size to maintain event history for a particular set of queries at a particular input rate and output rate, while event history that is required to maintain in a CEP system is much related to query depths of deployed queries in the system. The more query depth of a

active query has, the history of more events are stored in the memory in a CEP system, therefore the higher the ratio of live objects in the heap is. Considering the significant performance influence of query depth factor demonstrated and discussed in Section 5.2, the ratio of live objects in the heap can be applied to implicate the performance of query depth factor in CEP systems.

There are many performance tools provide light-weight memory sampling containing information of live objects, e.g., VisualVM and NetBeans[2]. However, the unreachable objects can not be traced through this approach, but through heap dump. Since the generation and analysis of heap dump requires memory itself, getting number of unreachable objects is costly in performance, especially in real-time performance monitoring and analysing. Therefore, to avoid costly heap dump in the system, the calculation of the ratio of live objects in the heap in Formula 5.5 can be transformed to Formula 5.6:

$$\gamma = \frac{B_{live}}{B_{heap}} \tag{5.6}$$

where B_{live} is bytes of live objects in the heap, and B_{heap} is bytes of used heap. The real-time information about the bytes of live objects in the heap can be traced by memory sampling. Real-time information about the bytes of used heap can be obtained from performance tools, such as VisualVM and NetBeans.

5.4.4 *Summary of Findings*

The performance impact in a CEP system from garbage collection is studied and presented. High garbage collection activity results in worst response time and input throughput. Although garbage collection is another vast research area, this performance study in CEP systems shall raise the practical performance management issues in a CEP platform with dynamic memory management.

The ratio of live objects in the heap as a performance metric is proposed in measuring CEP system performance. The ratio of live objects in the heap is more precise in indicating the available heap after gar-

2 NetBeans Profiler.NetBeans Profiler

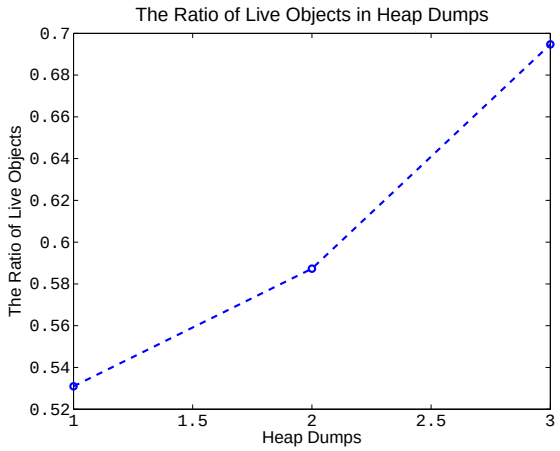


Figure 5.31: The ratio γ of the number of live objects in the heap

bage collection is performed. Furthermore, it can be used to implicate the performance of query depth factor in CEP systems. Therefore, the ratio of live objects can be applied as a fundamental performance metric in performance management of CEP systems.

5.5 SUMMARY

In this chapter, we study response time measurement, the load of and the complexity of queries that are deployed and memory management in CEP systems based on the CEPBen on Esper. Three new metrics are proposed in the studies. Firstly, response time of targeted event is proposed as a performance metric representing CEP system users' perception of response time. Secondly, maximum query load is proposed as a capacity indicator for CEP systems. And thirdly, number of live objects in heap as a performance metric for heap usage with experiments, evaluation and implementation of these metrics. Besides, query depth is proposed to describe the complexity of queries and presented the performance impact that it causes on CEP systems.

6

CONCLUSIONS AND FURTHER WORK

This thesis focuses on the performance management of complex event processing systems. Some challenges that complex event processing systems face in performance management are highlighted in Chapter 1. To tackle these challenges, a benchmark platform for CEP systems, CEPBen, is developed to provide a flexible environment for exploring performance factors and metrics in CEP systems. The benchmark is implemented on Esper event processing platform and used to explore performance factors and new metrics in performance management of CEP systems.

In this chapter, we will summarize the contributions of our work and discuss the direction of future research.

6.1 BACKGROUND RESEARCH

The background research of this thesis is described in Chapter 1 and Chapter 2. A general introduction to complex event processing systems and performance management of computer systems is presented in Chapter 1. Chapter 1 also highlights the challenges in performance management of CEP systems. Furthermore, Chapter 2 reviews the related research done in performance management of traditional transaction systems and CEP systems. The review mainly focuses on the existing benchmarks and applied performance metrics in performance management of these two types of systems.

Based on the literature review, it is concluded that new performance metrics and factors that distinguish CEP systems from the other types of systems, are not well explored. There are lacks of widely applied benchmarks for CEP systems and standards. Although a wide range of techniques are proposed to improve the performance of CEP systems, performance measurement and evaluation implemented in the demonstration of these new techniques are limited to the conventional measures, e.g., throughput. Very limited work is done in methodologies of performance management in event processing systems.

6.2 THE CEPBEN BENCHMARK PLATFORM

The CEPBen benchmark platform and its implementation are presented in Chapter 3 and Chapter 4. The CEPBen benchmark platform is developed to explore the fundamental functional performance of event processing systems: filtering, transformation and event pattern detection. A batched workload model is suggested to represent the workload of CEP engines. Response time, input throughput, output throughput and utilization are chosen to be the performance metrics. Performance factors including workload, query load, the depth that query statements perform on event history, garbage collection activity and machine configuration that CEP engines run on, are considered to be explored by this benchmark.

The CEPBen benchmark platform has the following features:

• It is able to present a varied workload to meet the requirements of different performance tests. Users can create events with their desired event properties, batch size, batch frequency.

• By setting the number of query statements and the depth of query statements, the benchmark presents varied degrees of application query complexity for investigating the system behaviours.

A performance-oriented framework is presented for implementing the CEPBen benchmark. Two groups of tests are designed and carried out. The results show that the tested CEP engine performs the best in filtering functionality, than transformation and pattern detection functionalities in response time and input throughput, while remaining high output throughput and consuming much less system resources. Event pattern detection has higher response time and higher input throughput than transformation, while transformation in performance tests has more output events and higher output throughput, and consumes less system resources than event pattern detection. After comparing our results with the performance study by Mendes et al. [18], it is concluded that CEP systems generally perform better in filtering than transformation and pattern detection.

6.3 PERFORMANCE METRICS AND FACTORS

The measurements of response time, query load, complexity of queries and memory management in CEP systems are investigated in Chapter 5. As a result, several new performance metrics and one performance factor are proposed and evaluated.

Traditionally response time is measured from the detection time of the last event that triggers the derived event to the generation time of the derived event in event pattern detection. However, event patterns can be used to catch a particular event that happens in a certain pattern. In such situation, the time that the system takes to respond to this particular event can be of interest to users. Thus, this response time of targeted event can be the users' perception of response time.

Response time of targeted event is defined as the time difference between the detection time of the targeted event and the time of the derived event when the event pattern is matched. The implementation of the response time of targeted event measurement is discussed. The implementation of measuring the response time of targeted event requires modifying the queries to mark the targeted event, so that the listener for the query can extract the detection time of the targeted event. Modification of queries does not change the structure of the queries, therefore, it does not change the system behaviours. The measurements of traditional response time and response time of targeted event with various workloads are compared and presented in Chapter 5.

Query depth is defined to describe the complexity of queries deployed in CEP engines. It is the number of primitive events that are used to produce a composite event or a derived event in a query statement. Our experiments shows that query depth significantly influences the performance of CEP systems. The system with queries that have less query depth has better response time and higher input throughput while outputs heavier load, comparing to the tested system with queries that have more query depth.

Query load is the number of active queries deployed in a CEP system. When the workload and query depth of queries are fixed, the more active queries a CEP system can support, the better capacity the CEP system has. Therefore, maximum query load can be used as an capacity indicator for event processing systems. Maximum query load of a CEP system can be affected by the workload and query depth of de-

ployed queries. The experimental results demonstrate that query load has significant impact on performance. The more the query load is, the higher response time that the system performs, the lower average input throughput that the system processes. Trade-offs between 99th percentile response time and between average input throughput and query load are also studied. The implementation of query load requires some programming on constructing a counter to record the newly deployed started queries and state changes on old deployed queries in a CEP system.

Number of live objects in the heap is proposed as a performance metric to reveal the usage of heap considering the effect that the garbage collection releasing the memory. Used heap contains both live objects and unreachable objects which will be collected when garbage collection happens. Therefore, the ratio of live objects in the heap can be used to indicate the heap usage in the near future after garbage collection occurs.

The ratio of live objects in the heap is especially useful because it is associated with query depth. It can be applied to indicate the performance of query depth factor in CEP systems. Initially, we define the ratio of live objects in the heap as the ratio of the number of live objects and the total number of objects (live and unreachable objects) in the heap. The number of live objects can be obtained by memory sampling from performance tools. However, the number of unreachable objects can only be acquired by heap dump which has negative impact on the performance of JVM. Therefore, we transform the calculation of the ratio of live objects in the heap to another way of calculation. The ratio of live objects can be represented as the ratio between the bytes of live objects and bytes of used heap.

6.4 FUTURE WORK

In this thesis, the CEPBen Benchmark platform is developed as a test bed for exploring performance characteristics of complex event processing systems in performing varying functionalities. The capability of this benchmark has been demonstrated in later implementation and experiments on Esper, a free open-source complex event processing platform. This benchmark can be reused to study the performance of CEP systems.

However, the real-time performance monitoring in the framework is very resource-consuming. In experiments which require more system resources, the real-time performance monitoring module has to be turned off. One way of improving the performance monitor is to develop a network socket module to transfer performance data to a stand-alone machine for performance analysis and display in real time.

Two types of query representation are used in the event processing engines today for event processing. One is variations of the standard conventional relational database query language: Structured Query Language (SQL), e.g. Event Processing Language (EPL) in Esper; the other query language is declarative language, e.g. Drools Fusion [90]. As the complexity of queries is reckoned as an influential performance factor in performance management, query depth is proposed regarding to the complexity of queries and tested on Esper to reveal the performance impact. Since the query language applied in Esper is a variation of SQL. In future, It would be of interest to investigate the complexity of queries which are written in declarative language.

Furthermore, the ratio of live objects in the heap cannot be obtained straightforward. It can be implemented and monitored at real-time by calculating the information gathered from some available performance tools. We propose to develop a plug-in for the existing performance tools for monitoring the ratio of live objects in the heap in CEP systems in real time.

The CEPBen benchmark platform can be applied in exploring more interesting factors and metrics. The proposed performance metrics and the methodology of studying performance of CEP functionalities in this thesis can be applied in performance management of CEP systems and benefit both developers and users of CEP systems.

Comparing performance of various CEP products are the interest of many developers and users in the CEP market. However, in this thesis we did not compare different CEP engine products. In the future, CEPBen can be implemented on other CEP engines. The performance results from these tests can be used for comparing CEP products.

BIBLIOGRAPHY

[1] O. Etzion and P. Niblett, *Event Processing in Action*. Stamford: Manning Publication Co., 2011. (Cited on pages 10, 14, 15, 18, 22, 45, 50, 52, 87, and 89.)

[2] C. Moxey, H. Lalanne, G. Sharon, M. Peters, M. Edwards, O. Etzion, M. Ibrahim, S. Iyer, M. Monze, Y. Rabinovich, and K. Stewart, "A conceptual model for event processing systems," *developerWorks in IBM*, Feb 2010. (Cited on pages 10, 15, and 19.)

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393–422, 2002. (Cited on page 14.)

[4] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. (Cited on page 14.)

[5] D. Yupho and J. Kabara, "Continuous vs. event driven routing protocols for wsns in healthcare environments," *Pervasive Health Conference and Workshops, 2006*, pp. 1 –4, nov. 2006. (Cited on page 15.)

[6] J. Bacon, A. R. Beresford, D. Evans, D. Ingram, N. Trigoni, A. Guitton, and A. Skordylis, "Time: An open platform for capturing, processing and delivering transport-related data," 2008, pp. 687–691. (Cited on page 15.)

[7] H. Wu, B. Salzberg, and D. Zhang, "Online event-driven subsequence matching over financial data streams," in *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 2004, pp. 23–34. (Cited on page 16.)

[8] J. Altmann, T. Bartha, and A. Pataricza, "An event-driven approach to multiprocessor diagnosis." 8th Symp. on Microcomputer and Microprocessor Applications, 1994, pp. 109–118. (Cited on page 16.)

[9] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam, "Critical event prediction for proactive management in large-scale computer clusters," in *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2003, pp. 426–435. (Cited on page 16.)

[10] D. A. Reed, R. A. Aydt, L. DeRose, C. L. Mendes, R. L. Ribler, E. Shaffer, H. Simitci, J. S. Vetter, D. R. Wells, S. Whitmore, and Y. Zhang, "Performance analysis of parallel systems: Approaches and open problems," pp. 239–256, 1998. (Cited on page 17.)

[11] J. Yan, "Performance tuning with aims-an automated instrumentation and monitoring system for multicomputers," vol. 2, jan. 1994, pp. 625 –633. (Cited on page 17.)

[12] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, "Monitoring streams: a new class of data management applications," in *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 2002, pp. 215–226. (Cited on pages 17 and 37.)

[13] A. Geppert, M. Berndtsson, D. Lieuwen, and J. Zimmermann, "Performance evaluation of active database management systems using the beast benchmark," Tech. Rep., 1996. (Cited on pages 17 and 39.)

[14] S. Kounev, B. J. Sachs, K., and A. Buchmann, "A methodology for performance modeling of distributed event-based systems." 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC), 2008. (Cited on pages 17, 31, 32, and 33.)

[15] R. Van Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Trans. Comput. Syst.*, vol. 21, no. 2, pp. 164–206, 2003. (Cited on page 17.)

[16] T. Osogami and S. Kato, "Optimizing system configurations quickly by guessing at the performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 1, pp. 145–156, 2007. (Cited on page 17.)

[17] T. Grabs and M. Lu, "Measuring performance of complex event processing systems," pp. 83–96, 2012. (Cited on pages 18 and 32.)

[18] M. Mendes, P. Bizarro, and P. Marques, "A performance study of event processing systems," in *Performance Evaluation and Benchmarking*, ser. Lecture Notes in Computer Science, R. Nambiar and M. Poess, Eds. Springer Berlin / Heidelberg, 2009, vol. 5895, pp. 221–236. (Cited on pages 18, 31, 33, 77, and 130.)

[19] A. Kozlenkov, D. Jeffery, and A. Paschke, "State management and concurrency in event processing," in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '09. New York, NY, USA: ACM, 2009, pp. 23:1–23:4. [Online]. Available: http://doi.acm.org/10.1145/1619258.1619289 (Cited on page 22.)

[20] J. C. Browne, "A critical overview of computer performance evaluation," in *ICSE '76: Proceedings of the 2nd international conference on Software engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, pp. 138–145. (Cited on page 24.)

[21] J. P. Buzen, "Fundamental laws of computer system performance," in *Proceedings of the 1976 ACM SIGMETRICS conference on Computer performance modeling measurement and evaluation*, ser. SIGMETRICS '76. New York, NY, USA: ACM, 1976, pp. 200–210. [Online]. Available: http://doi.acm.org/10.1145/800200.806196 (Cited on page 24.)

[22] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques For Experimental Design, Measurements Simulation And Modeling*. John Wiley & Sons, Inc., 1991. (Cited on pages 24, 26, 33, 47, 48, and 49.)

[23] P. Fortier and H. Michel, *CSPerformanceEvaluationPrediction*, ser. ISBN: 978-1-55558-260-9. Digital Press, June 2003. (Cited on pages 24, 26, and 27.)

[24] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. (Cited on pages 25 and 26.)

[25] K. B. Irani and H.-L. Lin, "Queueing network models for concurrent transaction processing in a database system," pp. 134–142, 1979. [Online]. Available: http://doi.acm.org/10.1145/582095.582116 (Cited on page 25.)

[26] A. Avritzer and E. J. Weyuker, "Deriving workloads for performance testing," *Software: Practice and Experience*, vol. 26, no. 6, pp. 613–633, 1996. (Cited on page 25.)

[27] E. Weyuker and F. Vokolos, "Experience with performance testing of software systems: issues, an approach, and case study," *Software Engineering, IEEE Transactions on*, vol. 26, no. 12, pp. 1147–1156, 2000. (Cited on page 25.)

[28] *TPC Benchmark C Standard Specification*, Revision 5.11 ed., Transanction Processing Performance Council (TPC), February 2010. [Online]. Available: http://www.tpc.org/tpcc/spec/tpcc_current.pdf (Cited on page 27.)

[29] *TPC Benchmark DS (TPC-DS): The New Decision Support Benchmark Standard*, Version 1.1.0 ed., Transaction Processing Performance Council (TPC), April 2012. [Online]. Available: http://www.tpc.org/tpcds/spec/tpcds_1.1.0.pdf (Cited on page 28.)

[30] *TPC Pricing Specification Standard Specification*, Version 1.7.0 ed., Transaction Processing Performance Council (TPC), Novermber 2011. (Cited on page 28.)

[31] *TPC-Energy Standard Specification*, Version 1.2.0 ed., Transaction Processing Performance Council (TPC), June 2010. (Cited on page 28.)

[32] *TPC BENCHMARK E Standard Specification*, Version 1.12.0 ed., Transaction Processing Performance Council (TPC), June 2010. [Online]. Available: http://www.tpc.org/tpce/spec/v1.12.0/tpce-v1.12.0.pdf (Cited on page 29.)

[33] *TPC BENCHMARK H Standard Specification*, Version 2.15.0 ed., Transaction Processing Performance Council (TPC). [Online]. Available: http://www.tpc.org/tpch/spec/tpch2.15.0.pdf (Cited on page 29.)

[34] *TPC Virtual Measurement Single System Standard Specification*, Version 1.1.0 ed., Transaction Processing Performance Council (TPC), Novermber 2012. [Online]. Available: http://www.tpc.org/tpcvms/spec/tpc-vms-v1_1_0.pdf (Cited on page 29.)

[35] T. L. Anderson, A. J. Berre, M. Mallison, H. H. Porter III, and B. Schneider, "The hypermodel benchmark," pp. 317–331, 1990. (Cited on page 30.)

[36] R. G. G. Cattell and J. Skeen, "Object operations benchmark," *ACM Trans. Database Syst.*, vol. 17, no. 1, pp. 1–31, Mar. 1992. [Online]. Available: http://doi.acm.org/10.1145/128765.128766 (Cited on page 30.)

[37] T. Böhme and E. Rahm, "Xmach-1: A benchmark for xml data management," *Datenbanksysteme in Büro, Technik und Wissenschaft*, pp. 264–273, 2001. (Cited on page 30.)

[38] S. Bressan, M. L. Lee, Y. G. Li, Z. Lacroix, and U. Nambiar, "The xoo7 benchmark," pp. 146–147, 2003. (Cited on page 30.)

[39] A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu, and R. Busse, "Xmark: A benchmark for xml data management," pp. 974–985, 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=1287369.1287455 (Cited on page 30.)

[40] M. Nicola, I. Kogan, and B. Schiefer, "An xml transaction processing benchmark," pp. 937–948, 2007. [Online]. Available: http://doi.acm.org/10.1145/1247480.1247590 (Cited on page 30.)

[41] O. Council, *APB-1 OLAP Benchmark*, release ii ed., OLAP Council, November 1998, available at: http://www.olapcouncil.org/. (Cited on page 30.)

[42] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith, "The sequoia 2000 storage benchmark," *SIGMOD Rec.*, vol. 22, no. 2, pp. 2–11, Jun. 1993. [Online]. Available: http://doi.acm.org/10.1145/170036.170038 (Cited on page 30.)

[43] J. Gray, "Benchmark handbook: for database and transaction processing systems," 1992. (Cited on page 30.)

[44] A. B. Chaudhri, "An annotated bibliography of benchmarks for object databases," *SIGMOD Rec.*, vol. 24, no. 1, pp. 50–57, Mar. 1995. [Online]. Available: http://doi.acm.org/10.1145/202660.202668 (Cited on page 30.)

[45] E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams," 2006. (Cited on pages 31 and 34.)

[46] A. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, W. M. White *et al.*, "Cayuga: A general purpose event monitoring system," 2007. (Cited on page 31.)

[47] N. P. Schultz-Møller, M. Migliavacca, and P. Pietzuch, "Distributed complex event processing with query rewriting," p. 4, 2009. (Cited on page 31.)

[48] G. T. Lakshmanan, Y. G. Rabinovich, and O. Etzion, "A stratified approach for supporting high throughput event processing applications," in *DEBS '09: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*. New York, NY, USA: ACM, 2009, pp. 1–12. (Cited on pages 31 and 34.)

[49] *Oracle Complex Event Processing Performance*, Oracle, November 2008. (Cited on pages 32, 45, and 86.)

[50] K. Isoyama, Y. Kobayashi, T. Sato, K. Kida, M. Yoshida, and H. Tagato, "A scalable complex event processing system and evaluations of its performance," pp. 123–126, 2012. [Online]. Available: http://doi.acm.org/10.1145/2335484.2335498 (Cited on page 32.)

[51] M. R. N. Mendes, P. Bizarro, and P. Marques, "A framework for performance evaluation of complex event processing systems," in *DEBS '08: Proceedings of the second international conference on Distributed event-based systems*. New York, NY, USA: ACM, 2008, pp. 313–316. (Cited on pages 33 and 36.)

[52] S. Wasserkrug, A. Gal, O. Etzion, and Y. Turchin, "Complex event processing over uncertain data," pp. 253–264, 2008. (Cited on page 34.)

[53] S. Akram, M. Marazakis, and A. Bilas, "Understanding and improving the cost of scaling distributed event processing," pp. 290–301, 2012. [Online]. Available: http://doi.acm.org/10.1145/2335484.2335516 (Cited on page 34.)

[54] S.-K. Chen, J.-J. Jeng, and H. Chang, "Complex event processing using simple rule-based event correlation engines for business performance management," *E-Commerce Technology, 2006. The 8th IEEE International Conference on and Enterprise Computing, E-Commerce, and E-Services, The 3rd IEEE International Conference on*, pp. 3 –3, jun. 2006. (Cited on page 34.)

[55] Q. Chen, Z. Li, and H. Liu, "Optimizing complex event processing over rfid data streams," pp. 1442–1444, 2008. (Cited on page 34.)

[56] W. Xu, J. L. Hellerstein, B. Kramer, and D. A. Patterson, "Control considerations for scalable event processing," in *DSOM*, ser. Lecture Notes in Computer Science, J. Schönwälder and J. Serrat, Eds., vol. 3775. Springer, 2005, pp. 233–244. [Online]. Available: http://dx.doi.org/10.1007/11568285_20 (Cited on page 34.)

[57] W. Zheng, R. Bianchini, and T. D. Nguyen, "Automatic configuration of internet services," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 219–229, 2007. (Cited on page 34.)

[58] A. Saboori, G. Jiang, and H. Chen, "Autotuning configurations in distributed systems for performance improvements using evolutionary strategies," jun. 2008, pp. 769 –776. (Cited on page 34.)

[59] P. Bizarro, "Bicep benchmarking complex event processing systems." (Cited on pages 35 and 42.)

[60] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*, 1st ed. The MIT Press, June 1999. (Cited on page 36.)

[61] *Towards a standard event processing benchmark*, ser. ICPE '13. New York, NY, USA: ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2479871.2479913 (Cited on pages 36 and 47.)

[62] M. R. Mendes, P. Bizarro, and P. Marques, "Fincos: Benchmark tools for event processing systems," pp. 431–432, 2013. [Online].

Available: http://doi.acm.org/10.1145/2479871.2479941 (Cited on page 36.)

[63] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts, "Linear road: A stream data management benchmark." VLDB Conference, September 2004. (Cited on pages 37, 42, and 111.)

[64] Q. Yang and H. N. Koutsopoulos, "A microscopic traffic simulator for evaluation of dynamic traffic management systems," *Transportation Research Part C: Emerging Technologies*, vol. 4, no. 3, pp. 113 – 129, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/B6VGJ-3VWT8KB-1/2/4bfda91270dcf22f26439123b886be90 (Cited on pages 37 and 47.)

[65] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse, "The xml benchmark project," Amsterdam, The Netherlands, The Netherlands, Tech. Rep., 2001. (Cited on page 38.)

[66] K. Sachs, S. Kounev, J. Bacon, and A. Buchmann, "Performance evaluation of message-oriented middleware using the specjms2007 benchmark," *Performance Evaluation*, vol. 66, no. 8, pp. 410 – 434, 2009, selected papers of the Fourth European Performance Engineering Workshop (EPEW) 2007 in Berlin. [Online]. Available: http://www.sciencedirect.com/science/article/B6V13-4VDY7VS-1/2/d1e1af7baf5596cab667c40283ebbc81 (Cited on page 38.)

[67] K. Sachs, "Benchmarking and performance modeling of event-based systems," *it–Information Technology*, vol. 51, p. 5, 2009. (Cited on page 38.)

[68] A. Geppert, S. Gatziu, and K. R. Dittrich, "A designer's benchmark for active database management systems: 007 meets the beast," in *RIDS '95: Proceedings of the Second International Workshop on Rules in Database Systems*. London, UK: Springer-Verlag, 1995, pp. 309–326. (Cited on pages 38, 39, and 42.)

[69] D. J. Carey, M. J. abd DeWitt and J. F. Naughton, "The 007 benchmark," *SIGMOD Rec.*, vol. 22, no. 2, pp. 12–21, 1993. (Cited on page 39.)

[70] M. J. Carey, D. J. DeWitt, C. Kant, and J. F. Naughton, "A status report on the 007 oodbms benchmarking effort," in *OOPSLA '94: Proceedings of the ninth annual conference on Object-oriented programming systems, language, and applications*. New York, NY, USA: ACM, 1994, pp. 414–426. (Cited on page 39.)

[71] M. Berndtsson, "Acood: an approach to an active object oriented dbms," *Master's thesis, Department of Computer Science, University of Sk ovde*, 1991. (Cited on page 39.)

[72] C. Collet and T. Coupaye, "The naos system," vol. 24, no. 2, p. 474, 1995. (Cited on page 39.)

[73] R. Agrawal and N. H. Gehani, "Ode (object database and environment): the language and the data model," *SIGMOD Rec.*, vol. 18, no. 2, pp. 36–45, Jun. 1989. [Online]. Available: http://doi.acm.org/10.1145/66926.66930 (Cited on page 39.)

[74] A. Geppert, S. Gatziu, K. R. Dittrich, H. Fritschi, and A. Vaduva, "Architecture and implementation of the active object-oriented database management system samos," *University of Zurich*, 1995. (Cited on page 39.)

[75] "Oracle complex event processing exalogic performance study - an oracle white paper," 2011, available at: http://www.oracle.com/. [Online]. Available: http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/cepexalogicwhitepaperfinal-498043.pdf (Cited on pages 45, 86, and 99.)

[76] "Esper performance," 2007, available at: http://docs.codehaus.org/. [Online]. Available: http://docs.codehaus.org/display/ESPER/Esper+performance (Cited on pages 45 and 86.)

[77] R. Tibbetts, "Performance & scalability characterization," available at: http://www.streambase.com. [Online]. Available: http://www.streambase.com/wp-content/uploads/downloads/StreamBase_

White_Paper_Performance_and_Scalability_Characterization.pdf (Cited on pages 45 and 86.)

[78] D. C. Luckham, *Event Processing for Business: Organizing the Real Time Enterprise*. John Wiley & Sons (, 2011. (Cited on pages 48 and 89.)

[79] *Esper Reference Documentation*, version 5.0.0 ed., EsperTech Inc., 2014. (Cited on pages 63 and 81.)

[80] R. F. Berry, J. L. Hellerstein, J. Kolb, and P. VanLeer, "Choosing a service level indicator-why not queue length?" in *Int. CMG Conference*, 1991, pp. 404–413. (Cited on page 101.)

[81] L. Ding, S. Chen, E. A. Rundensteiner, J. Tatemura, W.-P. Hsiung, and K. S. Candan, "Runtime semantic query optimization for event stream processing," pp. 676–685, 2008. (Cited on page 103.)

[82] X. Wang, K. S. Candan, and J. Song, "Complex pattern ranking (cpr): evaluating top-k pattern queries over event streams," pp. 279–290, 2011. [Online]. Available: http://doi.acm.org/10.1145/2002259.2002296 (Cited on page 103.)

[83] S. Babu and J. Widom, "Continuous queries over data streams," *SIGMOD Rec.*, vol. 30, no. 3, pp. 109–120, 2001. (Cited on page 111.)

[84] *Oracle JRockit JVM Diagnostics Guide*, R27.6 ed., Oracle, April 2009. (Cited on page 118.)

[85] P. R. Wilson, "Uniprocessor garbage collection techniques," pp. 1–42, 1992. (Cited on pages 118 and 122.)

[86] M. Hertz and E. D. Berger, "Quantifying the performance of garbage collection vs. explicit memory management," *SIGPLAN Not.*, vol. 40, no. 10, pp. 313–326, Oct. 2005. [Online]. Available: http://doi.acm.org/10.1145/1103845.1094836 (Cited on page 118.)

[87] Oracle, "Java garbage collection basics," *Available at: http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html*. (Cited on page 119.)

[88] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles, "Dynamic storage allocation: A survey and critical review," pp. 1–116, 1995. (Cited on page 122.)

[89] A. Reitbauer, K. Enzenhofer, A. Grabner, M. Kopp, S. PierZchala, and S. Wilson, *Java Enterprise Performance*. Compuware Corporation, 2012. (Cited on page 124.)

[90] *Drools Documentation*, Version 6.0.0.final ed., The JBoss Drools team, November 2013. (Cited on page 133.)