A Framework for Performance Evaluation of Complex Event Processing Systems

Marcelo R. N. Mendes
CISUC, University of Coimbra
Dep. Eng. Informática – Polo II
Univ. de Coimbra, 3030-290 Coimbra, Portugal
+351 239790000
mnunes@dei.uc.pt

Pedro Bizarro
CISUC, University of Coimbra
Dep. Eng. Informática – Polo II
Univ. de Coimbra, 3030-290 Coimbra, Portugal
+351 239790000
bizarro@dei.uc.pt

Paulo Marques
CISUC, University of Coimbra
Dep. Eng. Informática – Polo II
Univ. de Coimbra, 3030-290 Coimbra, Portugal
+351 239790000
pmarques@dei.uc.pt

ABSTRACT

Several new Complex Event Processing (CEP) engines have been recently released, many of which are intended to be used in performance sensitive scenarios - like fraud detection, traffic control, or health care systems. However, there is no standard means to assess the performance of a CEP engine. This omission is all the more relevant as there are currently many competing products, languages, architectures, data models, and data processing CEP techniques. A performance evaluation framework can help identify good design decisions and assist in improving engines. Here we demonstrate our work in progress: FINCoS, a framework that can be used to benchmark CEP systems. The proposed framework has five relevant characteristics:

i. Flexible (e.g., it allows changing the workload on the fly to measure reactions to peak loads);
ii. Independent of particular workloads;
iii. Neutral (not bound to any specific CEP product);
iv. Correctness check (validators can be plugged into the framework on demand to verify results);
v. Scalable (many of its components, like event generators, can be centrally orchestrated and run in parallel).

Note that the framework does not include a benchmark specification. In fact, it was designed such that diverse datasets and query scenarios can be easily attached and tested on several CEP engines.

As such, this framework has three key benefits: first, it can be used by the CEP community to more quickly devise and experiment new benchmarks for event processing systems. Second, CEP vendors can employ the framework in conjunction with their own test datasets to benchmark their systems internally. Finally, customers can use it with their real data and select the CEP engine that best fits their needs.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques

General Terms

Design, Performance, Measurement.

Keywords

Complex Event Processing, Performance Evaluation, Framework.

1. INTRODUCTION

Complex Event Processing (CEP) is a relatively new technology for processing and analyzing multiple events from distributed sources, with the objective of extracting useful information from them. It has been employed in diverse areas such as Business Activity Monitoring, fraud detection and network management, to perform tasks that vary from simple event correlation to detection of complex patterns of events or causality analysis. However, up to now, no effective method for comparing the performance and scalability of CEP engines has been established.

This omission is particularly relevant since there are currently many competing products, each with their own languages, architectures, data models, and data processing CEP techniques. A performance evaluation framework can help identify good and bad design decisions which in turn can assist both in the definition of standards and production of enhanced engines.

However, considering the present stage of the technology, benchmarking Complex Event Processing systems faces a series of challenges:

- Lack of standards – currently there are no standards in terms of query languages, data formats, semantics or terminology, which makes difficult to specify precisely the workload and the interfaces between the benchmark and CEP engines;
- Multiple domains of applications – CEP has been applied in many distinct fields, each one with its own specific requirements. This means that it will likely be necessary to design more than one workload and dataset in order to represent the diverse scenarios;
- Metrics – besides commonly used metrics like throughput or response time, assessing a CEP engine involves measuring other dimensions such as correctness of results (many possible correct answers, due to, for example, different event arrival sequences), capacity to adapt to variations in the load, (which tend to be frequent in event-processing systems) or

possibly precision and recall (to deal with uncertainty or fuzzy patterns, for instance);

In next section we describe how FINCoS, the proposed framework addresses some of those issues. Related work is discussed in Section 3.

1.1 The BiCEP project

This demonstration is the first outcome of the BiCEP project, whose final goal is to identify the core CEP requirements and develop benchmarks that allow an objective comparison of products and algorithms in spite of their architectural and semantic differences [2][3].

The main contribution of the FINCoS framework is to provide a flexible and neutral approach for experimenting diverse CEP systems, where multiple datasets, queries, answer validators, and engines can be easily attached, swapped, reconfigured and scaled.

2. THE FINCoS FRAMEWORK

Specifying a CEP benchmark when standards, applications and capabilities of this evolving technology are not well defined is a challenging task. Instead, we developed a functional but flexible tool, keeping it as generic as possible in such a way that workloads or CEP products have little or no impact in the framework itself. The objective is to use it to experiment varying combinations of datasets and queries, in order to identify the most relevant aspects for the definition of a benchmark for Complex Event Processing systems.

2.1 Architecture and Operation

Figure 1 illustrates the general structure of our performance evaluation framework, which includes five main components:

1. **Driver**[1] – simulates external sources of events; it is responsible for submitting load to the system under test (SUT). The events which compose the workload can be generated by the Driver itself according to user's specification or they can be loaded from a third-party file. Currently, the framework supports only simple directives for data generation, so it is likely that one needs to use its own dataset obtained directly from real applications or from simulations;
2. **Sink** – receives output events resulting from the queries running on CEP engines. The results are stored in log files and/or transmitted over network for subsequent validation;
3. **Controller** – it is the interface between the framework and the user. The Controller application is used to configure the setup of the environment (e.g., number of drivers and sinks or how many machines are used) and to control the other applications during performance tests (e.g., start and interrupt components, increase/decrease load intensity or change workload parameters);
4. **Adapters** – there is no standard event representation across CEP engines, so typically each product has its own set of supported formats. In the absence of a common format to all engines, we decided to use a neutral comma-separated-value (CSV) event representation and custom adapters to make the conversions to a format compatible with their corresponding CEP engine. We used the CSV format due to its simplicity and low overhead processing (experiments with alternative formats such as plain Java objects and name-value pairs showed a much higher utilization of resources when compared to CSV). We have implemented a few adapters for some products, and it should be easy for other people to extend that list – that is especially the case for some vendors whose products already support, fully or partially, event exchange using CSV format. The typical structure and functionality of an adapter is described in Subsection 2.2;
5. **Validator** – validates the results produced by CEP engines. It takes information from all drivers and all sinks and produces summaries indicating how well the SUT has performed. Those reports can be displayed while the tests are running or only after completion. A typical report includes performance metrics such as response time, total count of processed events and output events or average throughput, as well as information about the correctness of the results. Validators are query-specific and as such must be dynamically developed and attached to the benchmark suite.

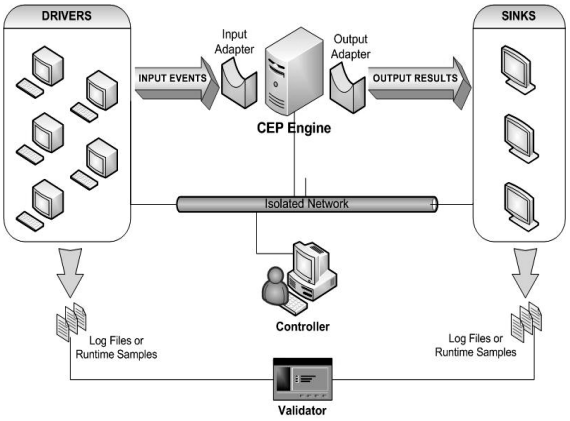

Figure 1 – Architecture of the Framework

Tests can be configured to use multiple Drivers and Sinks distributed over different machines. This architecture permits to increase the load over the SUT by scaling up the number of components when one or more Driver/Sink are in their limit.

The framework also provides flexible experiments setup. For instance, each Driver has its own workload (i.e., one or more datasets and event submission rates) and executes independently from other Drivers – which can be useful to simulate events coming from distinct sources. Of course, all Drivers can have exactly the same configuration (for instance, to increase the load over the SUT as discussed before). Moreover, the workload of a Driver can be made very dynamic, either during test setup, by dividing its execution in one or more sequential phases, with each phase having its individual workload characteristics (event rate,

[1] In CEP terminology, an entity that sends events is usually denominated 'event source' or 'producer' or still 'emitter'. The term 'driver' is often used in the benchmarking field to indicate a piece of software designed to put load on the system under test. We consider the latter term more appropriate because it emphasizes the fact that this component is a simulated event source and also includes extra functionality, like data generation and event scheduling.

duration and dataset) or by altering some workload parameters on-the-fly, while tests are running. The possibility to vary workload over time is useful for testing the ability of CEP engines to adapt to changes.

Finally, the framework was designed to be portable across different CEP products and test scenarios. That is achieved in two ways: first, by isolating the parts for which there is no standardization, namely, adapters and Validators, and making them "plug-and-play" components, which are developed and attached to the framework on demand; second, the use of replaceable Validators and the possibility of employing external datasets ensure that the framework is independent of particular workloads.

2.2 Adapter Structure

In the previous subsection we briefly explained how adapters are employed as mediators between the framework and different CEP products, by performing conversions between a CSV representation of events to vendor-specific native format. Here we describe the interface and behavior that an adapter should exhibit in order to be compatible with the framework.

Adapters receive events from Drivers as textual CSV messages, using plain sockets on a given local port. The message contains event's data as payload and an additional property indicating event's type. The latter is used by adapters to forward the event to the appropriate input stream on the CEP engine, after having done the required conversions. Likewise, resulting events from the CEP engine are delivered using plain sockets on a remote port of appropriate Sinks. Notice that adapters need to associate output streams to Sinks in order to deliver resulting events to the right destination (in CEP products this functionality is normally incorporated in their design tools). We have then built a small application that permits to specify these mappings between output streams and Sinks. The format used in delivery of output events is the same as input events, that is, CSV text messages, with event's data as payload and a property indicating event's type or the output stream where it came from.

Figure 2 illustrates our implementation of input and output adapters:

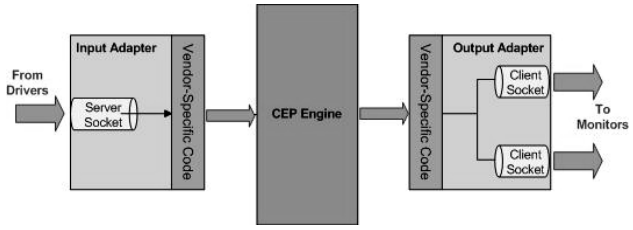

Figure 2 – Sample structure of adapters

We point out that only the vendor-specific part needs to change. This requires implementing simply two functions: one to send events and another to receive. The conversion between CSV records and product's internal representation is encapsulated inside those two functions.

Besides the conventional adapter, which provides direct connection to CEP engines, the framework also includes an adapter for JMS-based middlewares. This JMS adapter is useful for evaluating the performance of CEP engines under a very common configuration, namely when events are exchanged with external systems via messaging systems. Unlike the conventional adapter, the JMS adapter uses standard JMS map messages as its event representation format and JMS topics as intermediaries to CEP engines. In this way, the incorporated JMS adapter can be transparently used across different CEP products providing that they support integration with JMS sources. Figure 3 shows a sample test configuration using JMS:

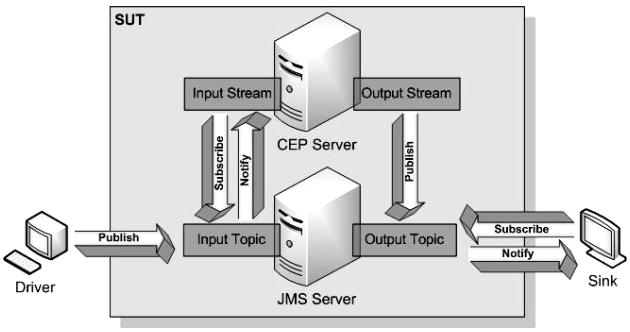

Figure 3 – Test configuration with a messaging infrastructure

2.3 Validation

As we mentioned before, validation does not have a universal logic: response time (and other metrics) can be measured in several different ways and how accuracy of the results is verified greatly depends of the queries executed during tests. For this reason, validation is not directly provided by the framework and should be performed by standalone components. Nonetheless, we have implemented a few sample Validators to measure response time and throughput as well as to perform correctness check for some common kinds of queries. Following, we show how validation is performed by those Validators and present alternatives to the approaches that we have chosen.

- Response Time Measurement – response time can be measured in two distinct ways: in real time, during the performance run, or only after the test completion. The first approach provides feedback sooner, which allows the user to better control the load during tests but it may overload the validation infrastructure. Currently we compute and display on-line estimates of response time at a reduced cost using sampling. Drivers can be configured to forward a small portion of its outgoing events to a Validator tool. Similarly, Sinks can also be configured to forward a fraction of the events it receives from the CEP engine to the Validator. The second approach consists in taking complete information from log files produced by all Drivers and all Sinks; upon test completion, the response time is accurately computed over the whole result set. Notice that in either case, the computation of response time is done in the same way: by inspecting the causal vector of each output event – which contains the ID's of the input events that caused it – and subtracting the timestamp of the last input event from the timestamp of the output event. For online measurement, however, there is an additional challenge of maximizing the matching between the sampled events from Sinks and those obtained from Drivers;
- Correctness Validation – another issue is how correctness of results should be checked by Validators. We envisage two approaches. The first is to compare, event by event, the result

produced by the CEP engine with the expected output (there may be more than only one correct output). A problem with this approach is that, depending on volume of events produced and test duration, validation can take too long to complete – more precisely, determining the expected output(s) may be a time-consuming operation. The second approach is to have a priori information about data. For instance, for a query that computes moving averages of some stock quote, it could be ensured that the test data will produce a result of, say, 50. For pattern detection queries, events could be generated in a top-down way, from the higher-level events to the raw events. For instance, it would be possible to stipulate constraints like "pattern X, which consists in event A followed by event B will happen N times in this dataset. Then, a summary is computed for output data and, compared with the expected result, which is known a priori. This approach has the advantage of being faster but it is potentially less accurate and is feasible only when using synthetic datasets. Moreover, it requires that data generation offers more sophisticated directives. For benchmarking purposes, we consider the first approach the most appropriate while for casual performance evaluations the second one should be enough;

- Adaptivity Assessment – although the framework enables testing the ability of a CEP engine to adapt to changes in the workload we have not defined a specific metric to assess this capability. Currently, we use other metrics and associate them with the moments when intentional changes in the workload occur (e.g. observing response time or throughput during and immediately after induced peak-load periods). We intend, however, to express adaptivity in terms of more precise metrics soon.

2.4 Demonstration Outline

A live software demonstration of our work will include:

- Test setup – configuration of Drivers and Sinks; creation of datasets using data generation component and loading of external files; scaling up the number of components;
- Sample performance runs in two or more CEP products, to illustrate the use of adapters;
- Dynamic changes in workload, to show the use of Controller application during tests;
- Validation and real time performance observations.

3. RELATED WORK

Previous work addresses the problem of performance evaluation in areas related to CEP. Berndtsson et al [1] present the BEAST benchmark for active databases systems. Arasu et al [5] describe Linear Road, a benchmark for Stream Data Management Systems. Finally, Sachs et al [4] introduce the SPECjms benchmark for JMS-based messaging systems. There are two main differences between our framework and those works: first, previous works are not intended to address the specific requirements of a benchmark for Complex Event Processing; second, all the aforementioned works include a benchmark specification, while our demonstration is about a performance evaluation framework, which can be used to devise new benchmarks.

4. SUMMARY

In this demonstration we presented FINCoS, a framework for performance evaluation of Complex Event Processing systems. Up to now, people who wish to assess the performance of CEP engines would have to build their own application that creates and submits a synthetic workload to those systems. With the framework, all that is necessary is to specify a few workload parameters, such as event submission rates or datasets (which can be created by the framework or loaded from external files). More elaborate test configurations can also be created, for example, by running Drivers and Sinks in parallel or by dynamically plugging Validators to check output results. We believe that the framework here described represents an important contribution to experimenting CEP systems and it will be helpful for accelerating improvements on engines as well the development of new benchmarks. The FINCoS framework is available at the BiCEP research project Web site [2].

5. REFERENCES

[1] Berndtsson M., Geppert A., Lieuwen D., Roncacio, C.: Performance Evaluation of Object-Oriented Active Database Management Systems Using the BEAST Benchmark. In Theory and Practice of Object Systems, v.4 n.3, p.135-149, 1998

[2] BiCEP research project Web site: http://bicep.dei.uc.pt.

[3] Bizarro, P.: BiCEP - Benchmarking Complex Event Processing Systems. In the Proceedings of Dagstuhl Seminar 07191 "Event Processing" November 2007. Available at: http://drops.dagstuhl.de/portals/index.php?semnr=07191

Accessed: March 2008.

[4] Sachs K., Kournev, S., Bacon, J., Buchmann, A.: Workload Characterization of the SPECjms2007 Benchmark. Available at: http://www.spec.org/workshops/2007/austin/papers/Designing_workload_Scenario_MOM.pdf

Accessed: March 2008.

[5] Arasu, A., Cherniack, M., Galvez, E., Maier, D., Maskey, A.S., Ryvkina, E., Stonebraker, M., Tibbetts, R.: Linear Road: A Stream Data Management Benchmark. In Proceedings of the 30th VLDB Conference (Toronto, Canada, 2004)