Towards a Standard Event Processing Benchmark

Marcelo R. N. Mendes
CISUC, University of Coimbra
Dep. Eng. Informática – Pólo II
Coimbra, Portugal
mnunes@dei.uc.pt

Pedro Bizarro
CISUC, University of Coimbra
Dep. Eng. Informática – Pólo II
Coimbra, Portugal
bizarro@dei.uc.pt

Paulo Marques
CISUC, University of Coimbra
Dep. Eng. Informática – Pólo II
Coimbra, Portugal
pmarques@dei.uc.pt

ABSTRACT
There has been an increasing interest both in academia and industry for systematic methods for evaluating the performance and scalability of event processing systems. A number of performance results have been disclosed over the last years, but there is still a lack of standardized benchmarks that allow an objective comparison of the different systems. In this paper, we present our work in progress: the BiCEP benchmark suite, a set of workloads, datasets and tools for evaluating different performance aspects of event processing platforms. In particular, we introduce *Pairs*, the first of the BiCEP benchmarks, aimed at assessing the ability of CEP engines in processing progressively larger volumes of events and simultaneous queries while providing quick answers.

Categories and Subject Descriptors
C.4 [Performance of Systems]: Measurement techniques

General Terms
Design, Experimentation, Measurement, Performance.

Keywords
Benchmark, CEP, Event Processing, Performance, Scalability.

1. INTRODUCTION
Even-driven applications are becoming increasingly prevalent in the most diverse domains of industry, such as capital markets, telecom, healthcare, sensor networks, and many others [5]. Most of these applications, however, are mission-critical or performance-sensitive, generating huge amounts of data and requiring very short response times. In order to guarantee that these applications have their requirements met, systematic methods for evaluating the performance of their enabling technologies are required. Aware of this need, SPEC has released the first industry standard benchmark for evaluating message-oriented middlewares (MOM) – SPECjms2007 [7].

MOM platforms play a fundamental role in event-driven applications, by ensuring that events are reliably disseminated to the appropriate destinations. However, exchanging messages that carry the information about events occurrence is only one of the features required by event-driven applications. After message delivery, they still need to filter the incoming events, aggregate their data, correlate seemingly unrelated events, detect situations of interest and react to them. These operations are typically performed with the aid of another class of systems, the so-called *Complex Event Processing* (CEP) engines, for which there is currently no standard benchmarks. In fact, a great part of the known performance numbers come from tests and studies conceived or sponsored by vendors (e.g., [8], [9]), many of which disclosed without the necessary details for replicating the results. A few others (e.g., [3], [6]), though neutral and detailed, used simple workloads and datasets that do not fully represent the typical usage of CEP engines. In order to address this lack of systematic evaluation methods, novel benchmarks are required. These benchmarks need to be clearly specified and easily understood. More importantly, they must exercise the entire spectrum of features offered by event processing platforms in a realistic manner.

In this paper we propose a first step towards filling this gap. We introduce our work-in-progress, the BiCEP benchmark suite, and the first of its domain-specific benchmarks: *Pairs*. The *Pairs* benchmark is set on the capital markets environment and exercises a wide range of features commonly found in most event processing applications, including:

- Filtering, aggregation, and correlation of events;
- Detection of event patterns and trends;
- State maintenance;
- Large number of simultaneous queries (increasing with the system scale);
- Changing load conditions.

Pairs was designed to assess the ability of the CEP systems in processing increasingly larger number of continuous queries and event arrival rates while providing quick answers – three quality attributes equally important for an event processing engine. The benchmark was also designed to be fully customizable, so that users can carry out performance studies that resemble more closely their own environments.

2. DESIGN PRINCIPLES
Even though most event processing applications share some common characteristics (e.g., the need for automated and timely answers), the several CEP domains differ significantly in their functional and performance requirements. For example, users in the capital markets are generally very concerned about processing latency, as short response times represent competitive advantage. Thus, sub-millisecond latencies are typically expected

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICPE'13, April 21–24, 2013, Prague, Czech Republic.
Copyright © 2013 ACM 978-1-4503-1636-1/13/04...$15.00.

in the algorithmic trading domain and the shorter the system takes to react, the better. However, other applications, like fraud detection and traffic monitoring, are generally not so latency-sensitive, and tolerate response times in the order of few seconds or even minutes. Performance concerns in these cases might revolve around volume of data or queries state size.

These significant differences in the performance requirements of the application domains makes virtually impossible for a single benchmark, with a single metric, to be representative of the entire spectrum of applications and provide all the information required by its heterogeneous target audience. For this reason, we opted to devise BiCEP as a set of smaller, domain-specific benchmarks, each with its own workload, dataset and metrics. Our goal is that each benchmark will allow evaluating one or more aspects of event processing systems (e.g. latency, scalability with respect to number of queries and rules, storage efficiency, etc.). Besides measuring CEP engines in multiple ways, a benchmark suite has a number of other advantages:

- *Extensibility*: as more information about CEP use-cases become available, new tests can be added to the suite, making it more comprehensive.
- *Understandability*: users can more easily relate the individual domain-specific benchmarks to their real applications.
- *Configurability*: the focus of the evaluation can be controlled by selecting whether all tests will be executed or only a subset of them will be considered.
- *Fairness/Portability*: CEP engines differ considerably in their focus, capabilities and query languages. Having multiple tests allows a system that does not perform well on a given benchmark (or is not able to implement it at all), to showcase its capabilities on a different scenario.

The BiCEP benchmark suite is the ongoing result of years of research and analysis of real event processing use-cases and platforms. Throughout the rest of this paper we outline its first benchmark, *Pairs*, and discuss our plans for extending the suite.

3. THE *Pairs* BENCHMARK

In this section we provide a brief overview of the *Pairs* benchmark. Note that due to space limitations, only the main aspects of the benchmark are discussed. For a complete specification please refer to [2].

3.1 Application Scenario

The scenario for *Pairs* is an investment firm where a number of analysts interact with an enterprise trading system responsible for automating and optimizing the execution of orders in stock markets. Users of the system pose trading strategies which are continuously matched against live stock market data. The exercised trading strategies belong to a category broadly known in the financial domain as *statistical arbitrage* and consist in monitoring the prices of two historically correlated securities, looking for temporary digressions that indicate an opportunity to capitalize on market inefficiencies.

The general structure of the benchmark scenario, including the main entities and the corresponding cardinalities, is depicted in Figure 1. Per every stock market M, a number of symbols (100) are monitored by the system, from which half are known to be mutually correlated. Each of the users of the system manages exactly five strategies. The number of users per market ranges from five up to fifty, depending on the scale factor. In the simplest case (5 users), there will be 25 strategies, each defined over a unique pair of correlated symbols. On the limit, each pair of correlated securities is monitored by ten strategies of different users, each with its own parameters.

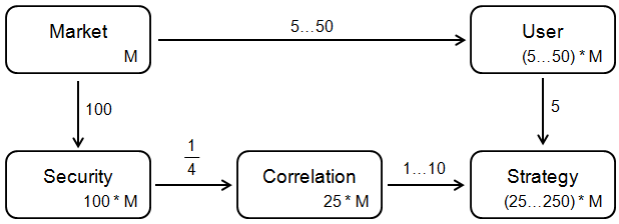

Figure 1: Entities and relationships in the *Pairs* benchmark

This structure allows evaluating not only if the tested system performs well on a multi-query scenario, but also its ability in sharing resources among similar queries.

3.2 Input Data

Input data for the *Pairs* benchmark consists in a stream of simulated stock market data with the following schema:

(*symbol*: **string**, *price*: **int**, *size*: **int**, *tickTS*: **long**, *TS*: **long**)

Each incoming tuple represents a trade operation executed in the stock market, such that *symbol* identifies the security being traded, *price* indicates the value in cents of the transaction, *size* represents the amount of shares negotiated, *tickTS* is the time, in milliseconds, at which the trade has been executed (i.e., simulation clock time) and *TS* is the actual time the record was sent to the CEP engine (i.e., wall clock time, added by the benchmark driver).

In the standard configuration, 2 hours of simulated market data are produced by a data generator application and submitted afterwards to the *system under test* (SUT). Tick arrivals follow a Poisson process [1], with its λ parameter – which represents the average arrival rate – varying over time. The reason for having a varying input rate is to simulate more realistically what happens in most real event processing applications, where new data arrives at different rates depending on the period of the day.

3.3 Workload

The benchmark workload consists in processing simultaneously a number of *Pairs* strategies. All strategies perform the same set of operations, described below, although using different parameters:

1. *Compute indicators*: the prices of a pair of securities are aggregated over a given time interval and then correlated to produce a *ratio*. The values of this ratio are then aggregated again, producing a moving average and upper and lower bands (these are usually referred as "Bollinger Bands").
2. *Signal opportunities*: the indicators produced in the previous step are used to determine possible opportunities to capitalize. This happens when the line formed by the values of the ratio crosses one of the bands.
3. *Position*: once a possible opportunity has been spotted, the system checks if it must change its current market position.

4. *Place orders*: if a change in market positioning is indeed required, the system must emit orders. This step involves identifying the appropriate values for the parameters of each order (i.e., size and price).
5. *Manage risk*: once a market position has been assumed, the system must detect if the prices keep drifting apart, countering the expected reversal trend, to prevent losses.

All the operations above are performed by each running strategy. Ideally the systems under test will be able to identify similarities among them and share resources during strategies execution.

3.4 Output

The output of the *Pairs* benchmark consists in two event streams: *Indicator* and *MarketOrder*. The former represents the output of the first step in the strategy execution process and is used in the benchmark scenario for visualization and auditing purposes (*the stream serves to produce a chart like Figure 2 that allows users to better understand the decisions taken by each strategy*).

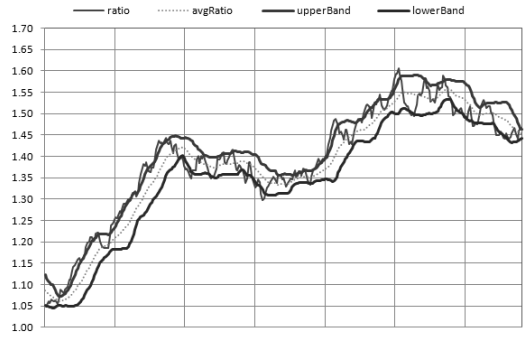

Figure 2: Chart showing the values of the *Indicator* stream

The second stream represents the orders that were issued as a result of the execution of the strategies. The schemas of the two output streams are shown below:

Indicator (		***MarketOrder*** (	
strategy	: **string**,	*strategy*	: **string**,
ratio	: **double**,	*type*	: **string**,
avgRatio	: **double**,	*symbol*	: **string**,
upperBand	: **double**,	*price*	: **int**,
lowerBand	: **double**,	*size*	: **int**,
inputTickTS	: **long**,	*inputTickTS*	: **long**,
inputTS	: **long**	*inputTS*	: **long**
)		)	

Tuples of the *Indicator* stream consist in a field *strategy*, indicating which strategy generated the result, and the fields *ratio*, *avgRatio*, *upperBand* and *lowerBand*, containing the values of the indicators. The *MarketOrder* stream consists in the fields *strategy*, again identifying the strategy that triggered the output, *type*, identifying the order as '*BUY*' or '*SELL*', and the fields *symbol*, *price* and *size*, which have the same meaning as in the input stream *StockTick*. Besides the payload, tuples from both streams include two timestamps: *inputTickTS* and *inputTS*. Both are derived from the input event that triggered the emission of the output tuple and represent respectively the tick occurrence time (simulation clock) and its arrival time (wall clock). The former is used for checking the correctness of the results while the latter is used for response time computation purposes.

3.5 Scaling

The scale factor (SF) in *Pairs* affects the number of users, and consequently the number of strategies executed in parallel as follows:

- Number of users: 5 SF
- Total number of strategies: 25 SF

Additionally, per every increment of ten in the scale factor, the input rate is incremented by 5,000 and the number of symbols is increased by 100 (this is to avoid too many similar strategies over the same symbols and to allow to observe how the system scales with changes in input rate and cardinality). The effect is as if a whole new market were now being monitored by a new team of analysts. For instance, for a scale factor of 15, there will be 75 users, each managing 5 strategies, on a total of 375 strategies running in parallel on the trading system, from which 250 are over the first set of 100 symbols and 125 are over the second set of 100 symbols.

3.6 Measures

As mentioned earlier, the purpose of the *Pairs* benchmark is to evaluate the ability of CEP systems in *processing increasing loads* while *providing quick answers*. Naturally, different users have different perceptions on the value of each dimension depending on their requirements (e.g. for some, the best system is simply the one that replies faster, while for others it is the one that handles more load). Nonetheless, in order to facilitate comparison among the several platforms and benchmark runs, we have defined a p_{score} metric to represent overall system performance[1]:

$$p_{score} = \frac{load}{99^{th} latency}$$

When defining the metric above, we tried to benefit systems that are able not only to process high volumes of events, but also react quickly and scale well with respect to the number of concurrent queries. Note, though, that the p_{score} exists essentially for comparison purposes, and that a *Pairs* report should always include a number of other measures (e.g., throughput, average and maximum latency, latency histogram, etc.) to help users to better understand the performance of the system under test and judge whether it fits their needs or not.

3.7 Is *Pairs* a Good Workload Scenario?

There are a number of reasons why we believe the *Pairs* benchmark represents a good test case for CEP platforms. First, the workload exercises several common features that appear repeatedly in most event processing applications: it *filters* out ticks from securities which are not of interest, *aggregates* events data over *temporal* and *count-based windows*, *correlates* price data for interrelated securities, *detects patterns* from price movements, keeps track and updates strategies' *state* upon the occurrence of certain events, and performs *lookups* to determine orders price and size. In addition, different from most benchmarks, which have a fixed set of queries, the number of queries in *Pairs* increases with the system size. This is in

[1] The term "*load*" in the formula is a function of the number of strategies and the input rate. Further information on how the p_{score} is computed can be found in the *Pairs* specification [2].

conformance with what happens in many real event processing applications and allows evaluating important aspects like query scalability and resource sharing.

Other key benefits of *Pairs* are understandability and representativeness. The benchmark mimics a niche of application where event processing platforms have perhaps been most successful – capital markets. In fact, most products use simple financial use-cases to exemplify the usage of their features and languages in their documentation, so in principle it should be easy for anyone reasonably familiar with CEP to understand *Pairs*. In addition, *Pairs* is loosely based on a real use-case, and as such has a good chance to be representative of its application domain.

Finally, *Pairs* allows a great deal of customization. Users can control load intensity by setting high-level workload parameters like input rate and number of simultaneous strategies, or by altering scenario characteristics such as number of securities and configuration of the strategies. While the results obtained from these "customized" runs cannot be compared to standard runs, the ability to customize the workload enables users to exercise the systems in a manner closer to their own real environment.

3.8 Implementation and Preliminary Results

The *Pairs* benchmark should be implemented and executed as illustrated in Figure 3 below:

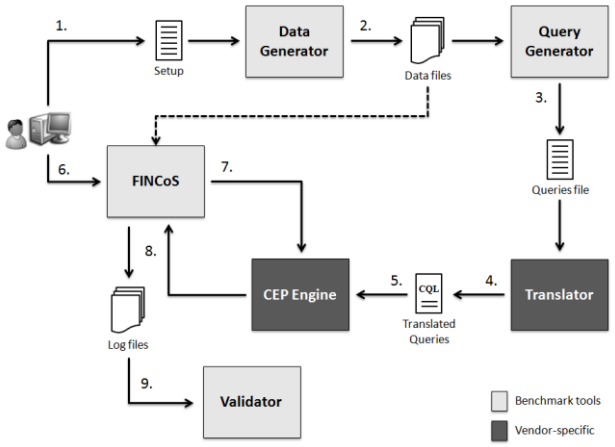

Figure 3 The execution flow of the *Pairs* benchmark

Initially, the user specifies a couple of workload parameters or, alternatively, uses the standard benchmark configuration to create a test setup (1). Then, a *data generator* application generates data and auxiliary files (2), which are used afterwards by a *query generator* to produce the strategies that compose the benchmark workload (3). The output of the query generator is then parsed by a vendor-specific *translator*, which converts the workload, initially represented in a neutral format (e.g., xml file), into the query language used by the SUT (4). After loading the query/rule set into the SUT (5), the user starts a performance run (6). During the run, the benchmark driver (*FINCoS*) loads the generated data file and submits the events on it to the SUT (7), which in turn returns the corresponding results to the framework (8). After test completion, a *validator* verifies the correctness of the answers produced by the SUT (9).

All the aforementioned tools are written in Java and require very little effort to be executed. The *data generator*, *query generator* and *validator* applications are specific to the *Pairs* benchmark and can be downloaded from [2]. The *FINCoS framework*, on the other hand, is benchmark-independent and can be found at [4].

We have recently implemented *Pairs* in a widely used open-source CEP engine and run some preliminary performance tests. In our experiments, the SUT managed to reach a maximum scale factor of 3, with processing latencies ranging from 5 up to 2,810 milliseconds (average: 33 ms; 99th-perc.: 106 ms), obtaining a p_{score} of 0.085. The tests also revealed some interesting facts about the system performance, like a regular increase of response time when faced with larger load levels and an odd variation on CPU utilization across the measurement interval. Further experiments and analysis are still required to fully understand these results, though. At the present moment, we are implementing the benchmark on another CEP engine to establish a basis for comparison.

4. SUMMARY

So far very little information about the performance of CEP engines has been made available, in spite of those systems being used in mission-critical or performance-sensitive scenarios. In this paper we delineate the path for filling this gap by introducing the BiCEP suite, and the first of its domain-specific benchmarks – *Pairs*. Both BiCEP and *Pairs* are work in progress, which we intend to further develop over the next months. In particular, we plan to extend the suite with new benchmarks as well as enhance *Pairs* and its tools. We promptly invite the community to contribute to this discussion.

5. ACKNOWLEDGMENTS

This research has been supported in part by the Portuguese Science and Technology Foundation (FCT), under grant Nº 45121/2008.

6. REFERENCES

[1] Aldridge, Irene, High-frequency trading: a practical guide to algorithmic strategies and trading. Wiley trading series. ISBN 978-0-470-56376-2.

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

[3] Dekkers, P. Master Thesis Computer Science. Complex Event Processing. Radboud University Nijmegen, Thesis number 574, October 2007.

[4] FINCoS Framework: https://code.google.com/p/fincos/

[5] Hinze, A., Sachs, K., Buchmann, A.P. Event-based applications and enabling technologies. In Proceedings of DEBS 2009.

[6] Mendes, M.R.N., Bizarro, P., Marques, P. 2009. A Performance Study of Event Processing Systems. In Proceedings of the 1st TPC Technology Conference (Lyon, France, August 2009), 221-236.

[7] SPECjsm2007 benchmark: http://www.spec.org/jms2007/

[8] STAC Report: Aleri Order Book Consolidation on Intel Tigertown and Solaris 10. Available at: http://www.stacresearch.com/node/3844

[9] White, S., Alves, A., Rorke, D. 2008. WebLogic event server: a lightweight, modular application server for event processing. In Proceedings of DEBS 2008, 193-200.