Benchmarking of Complex Event Processing Engine - Esper

Arun Mathew
123059007
Dept of Computer Science and Engineering
IIT Bombay
arunmathew@cse.iitb.ac.in

***Abstract*—Esper is an open source Complex Event Processing Engine. The project website has benchmarking details[1] for ESP[1] queries, for a 100 Mbit/s network. In this project we benchmark the CEP features of Esper. We measure the CPU utilization, memory utilization, latency and throughput for different kinds of CEP queries like conjunction, disjunction, sequence etc.**

I. Introduction

The existing metrics of Esper performance[1] are as follows

1) Esper exceeds 500,000 events per sec on a dual 2GHz CPU
2) less than 3us latency; less than 10us with 99% predictability
3) tops at 70 Mbit/s at 85% CPU usage
4) 1000 statements registered
5) linear scalability from 100 000 to 500 000 event/s on this hardware, with consistent results across different statements

But this result and benchmarking is from a set of ESP queries. i.e. queries which involved simple select queries and windowed aggregates on a single stream of data. Complex Event Processing is a super set of ESP. In CEP, we find patterns, derive new events based on a combination of input events, possibly from multiple streams of data. A good explanation of ESP vs. CEP can be found at [5].

Esper already provides a benchmarking kit as part of their example codes. The existing kit has a client which generates events of kind MarketData at a configured rate and sends it to a specified server IP at port 6789. MarketData event has generation timestamp(long), ticket(string), price(double), volume(int) as the attributes. Client generates subsequent event price and volume from a random number generator. The ticker range can be configured to N, then client will generate tickers from S0AA to S999 for N = 1000. The server part of the kit listens for multiple connections on 6789 and starts a ClientConnection thread for each connection. ClientConnection thread reads MarketData.SIZE/8 bytes from the SocketChannel and deserialize the MarketData event, marking it with the input time. ClientConnection then sends the event to the Esper Library/Engine using the cepProvider.sendEvent method. If the event triggers a match, then the appropriate subscriber method or updateListener method is called by the Esper engine.

[1]Event Stream Processing

Our objective is to benchmark the performance of Esper when CEP statements are registered with the engine. CEP queries and patterns often require some automation, which requires additional memory space to store the state and events. The processing time is also bound to increase as a new event has to be matched against any partial matches as well as for new automation.

In the following sections we describe the Esper engine, benchmarking setup, definition of parameters, the tabulated results, future work and conclusion.

II. Esper Architecture

We don't have a very detailed architectural diagram of Esper because the community doesn't release such materials to the public domain. The code is open source but only the APIs to use the engine are sufficiently documented. Hence, it is not a very developer-friendly open source project. Diagram 1 is what is published in the Esper Tech website.

Esper is written in JAVA, and works like a library to a JAVA application. We can send event objects to the Esper libary. We can register the query statements and their corresponding listeners or subscribers. Esper works like an inverted database. Event stream come into the engine and is run through the live queries. Queries can be windowed on time or length.

Esper accepts different event representations, POJO[2] events, java.util.Map events, object array events and XML events. Esper used EPL[3] to write SQL like statements to be run in the engine. Esper engine parameters can be tuned by the configuration xml file. Many parameters can be altered in the runtime as well using ConfigurationOperations object.

Esper also offers a commercial High Availability version called Esper HA. Esper HA has three additional modes (viz Durable, Resilient and Overflow) of operation which can be set on a per statement basis. Durable restarts the statements automatically in case the engine crashes/stops and is started again. Resilient stores the entire state of the query and hence enables continuing from the point the engine stopped.[3]

Esper HA documentation[4] claims small overhead compared to Esper to track changes. Throughput varies by -5% and memory/cpu profiles varies by +5%.

[2]Plain-Old Java Object
[3]Event Processing Language

Esper and Esper HA both work within a single node. i.e. their scalability is within a system based on the number of cores and amount of memory. Load balancing and clustering across a set of nodes have to be handled outside of Esper.

III. Benchmarking Setup

The setup and layout of systems are similar to the original benchmarking, except for the changes as stated below.

A. Modifications to the kit

We modified the Benchmarking Kit provided by Esper to suit our requirements. The main changes are as listed below

- unique ID for each event
- a new field 'test' to mark test events
- a new event type NewsData: with sentiment (double) as the attribute
- a pattern file for client: so that we can control the event sequence delivered to the Esper engine and hence have an expectation of the results. The pattern file allows specific values for each attribute and changing random generator seed for each attribute.
- client can emit configured number of events: by using "-count N" as the Client argument
- ticker range start and end can be specified
- server updateListener notes down latency
- client and server modified to transfer both MarketData and NewsData events over the same channel: this is done by reading the first 2 bytes and checking if the character is 'M' for MarketData and 'N' for NewsData. First field in the event is the uid, which starts with 'M' or 'N' according to the event type. The Server code then reads the remaining bytes according to the event type and then deserializes it.
- Increased the socket read buffer size and write buffer size: since time spent on read and write was increasing. Client writes 52 or 56 bytes at a time for NewsData and MarketData, respectively, and Server reads 2 bytes followed by 50 or 54. For best performance, we should be reading and writing large chunks at a time.

B. Pattern File

The pattern that we used for the current experiement is the following.

```
MarketData
NewsData
MarketData
NewsData
MarketData
MarketData
MarketData
MarketData
NewsData
NewsData
NewsData
NewsData
MarketData
MarketData
NewsData
MarketData
NewsData
MarketData
NewsData
```

C. System

Two KVM based VMs were created in a server with the following specifications
CPU : Intel(R) Xeon(R) CPU E5-2620 V2 @ 2.10GHz x 2 nos [12 cores x 2(Hyper Threading)]
Memory : 32G [8GiB DIMM DDR3 Synchronous 1600 MHz (0.6 ns) x 4 nos]
OS : Ubuntu 12.04.2 LTS Server
Network: Intel I350 Gigabit Network Connection

Server
CPU : QEMU Virtual CPU version 1.0 2GHz Pinned to one Core
Memory : 4GB allocated
OS : Ubuntu 12.04.2 LTS Server
Network: Realtek RTL-8139/8139C/8139C+ 100Mbit/s JVM Version : IcedTea6 1.12.6 64 bit JVM Options : -Xms1024m -Xmx1024m

Client
CPU : QEMU Virtual CPU version 1.0 2GHz Pinned to one Core
Memory : 1GB allocated
OS : Ubuntu 12.04.2 LTS Server
Network: Realtek RTL-8139/8139C/8139C+ 100Mbit/s JVM Version : IcedTea6 1.12.6 64 bit JVM Options : -Xms128m -Xmx128m

IV. Definition of Parameters & their control/evaluation

A. Latency

Latency is the time taken to detect a complex event since the last event in the set of triggering events are sent to the CEP engine. In our setup we note the time in milliseconds before sending each event. Upon matching a statement the updateListener function would be invoked with the events. There we update the stats module with the current time - last event time.

B. Throughput

Throughput is the maximum events per second which the CEP engine can process without loss of data or without clogging the queues. The current setup uses a channel which blocks writing, if the channel buffers are full. So the client program will not be able to write data to the channel any faster than the server consumes it. At 100% CPU utilization, the throughput may decrease a little and the latency may increase.

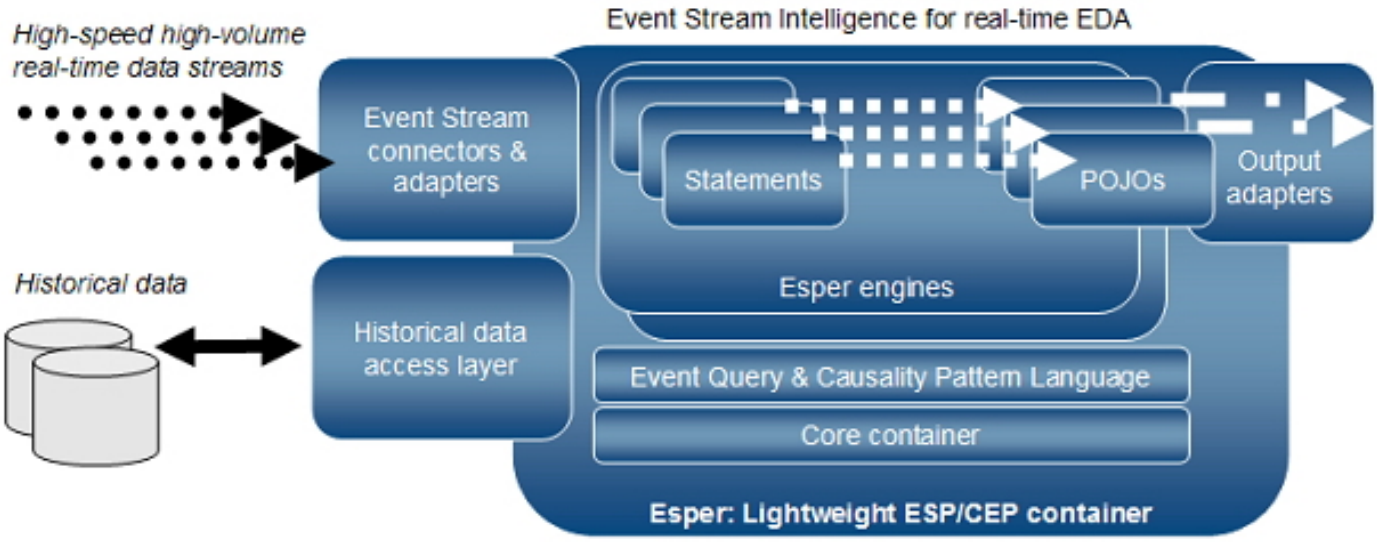

Fig. 1. Esper Architecture

C. CPU Utilization

It is the CPU Utilzation for different kinds of CEP query over different event rates for a given pattern. It was measured using a profiler called YourKit[2].

D. Memory Utilization

It is the memory profile for different kinds of CEP query over different event rates for a given pattern. It was measured using a profiler called YourKit[2].

E. Selectivity

This is the percentage of events that matches the predicates of a statement. This is controlled by adjusting the rate of events, ticker range and the pattern.

F. Number of Classes

This is the number of different types of events we use in a statement. We have to add new classes related to MarketData, say like NewsData, which contain certain sentiment value to the news articles on certain tickers, so that we can have some meaningful complex query to detect if a MarketData price decreased following a reduction in the sentiment value towards that ticker.

V. Experiment Design

We measure the throughput, latency, memory and CPU utilization for each of the specified queries to be evaluated under different event delivery rates.

We fixed the range of stock quotes being generated to 'S0AA' through 'S9AA' using the client setting -Desper.benchmark.symbol.start=0 -Desper.benchmark.symbol.end=9. The pattern for event generation is as specified above.

In our setup we run the runClient script with a rate N = 40, 400, 4,000, 40,000 & 400,000. For each rate N we send N * 10 events per class with test bit set to 1. This is to start the engine's processing and run it for some time allowing it to stabilize before delivering the actual events. The number of events per event class to be sent with test set to 1 is set by the property -Desper.benchmark.testeventcount=<N*10>. Then the client will send N * 10 events with test = 0. All our queries has test = 0 as a predicate. So it is in this 10 seconds that the events will be matched and the updateListener would be invoked.

The throughput & latency are logged from the Server process for each run. The CPU & Memory profiles are recorded from YourKit profiler. The samples are exported into a file and aggregated corresponding to each run.

The selectivity of each statement is specified in the queries table and is determined by the predicates in the statement and by the pattern. For example, selection query selects Market events with volume < 5. The value of volume is generated from 1-9 using a random number generator. So, the probability is 4/9. Now according to the pattern, half of the events are NewsData events, which means probability is 1/2 * 4/9. Again the experiment setup sends 2/3 of the events with test=1. So the final selectivity is 1/3 * 1/2 * 4/9 = 7.4%. Parameters for each query for each rate are evaluated thrice. There is an overhead of the profiling agent, which reduces the throughput slightly.

VI. Queries to be evaluated

The class of queries to be evaluated is listed in Table I. The operators are described in many of the survey papers. One good survey is from Anuj Thakkar's (IITB 2013 batch) MTP report.

VII. Results

The performance metrics for various classes are as in Table II. The throughput (figure 4) is found to be leveling at 120k events per second, with the CPU (figure 2) approaching saturation. The latency (figure 3) is found to decrease as the event rate increases.

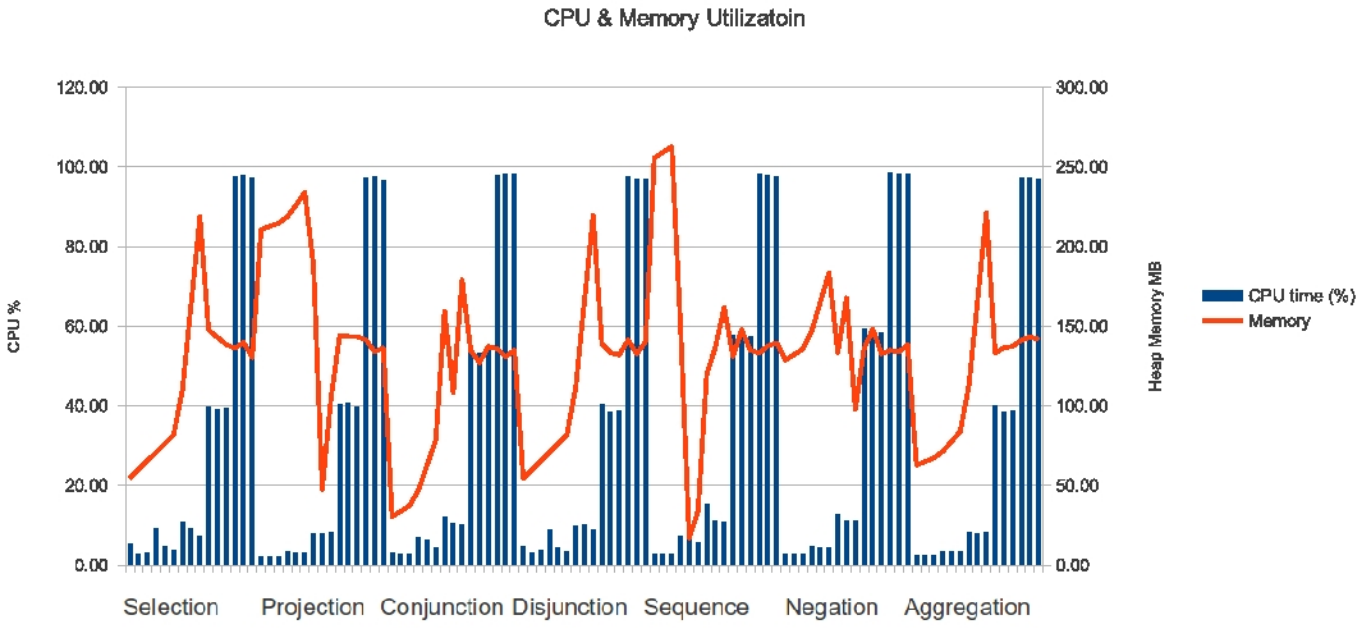

Fig. 2. CPU/Memory Utilization

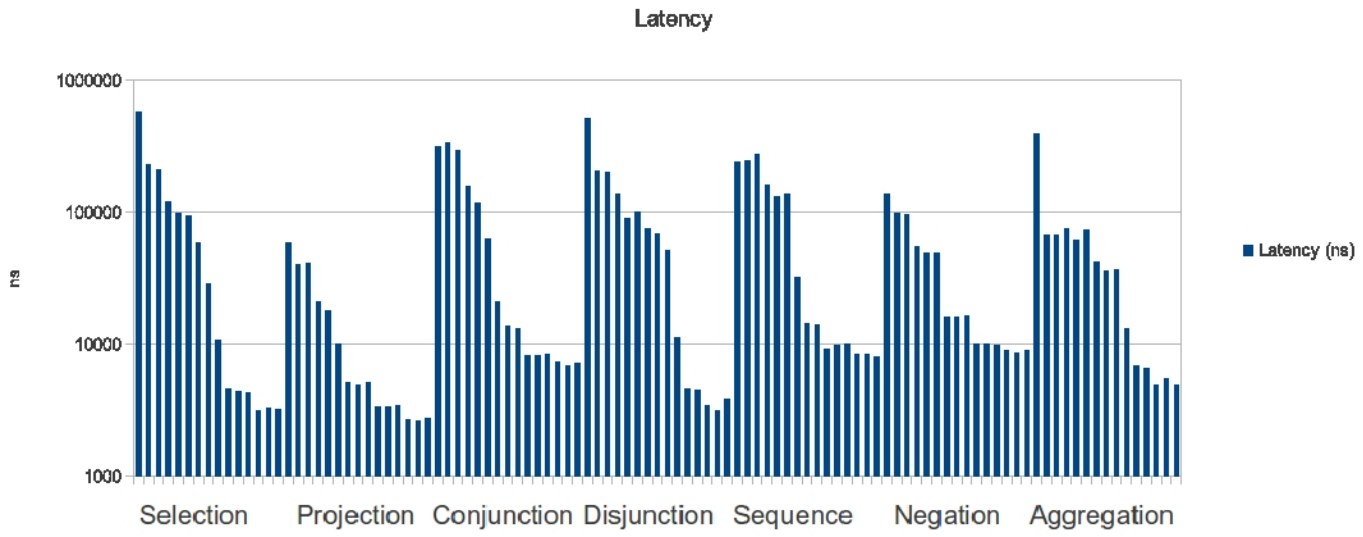

Fig. 3. Latency

VIII. Future Work

The future direction for this work would be to generalize the parameters and benchmarking steps. It could be made more modular, so that we can use the setup easily for other CEP engines. Similar figures for Esper HA in the durable, resilient and overflow modes can be evaluated. Also, the performance can be evaluated in a multi-core muti-threaded scenario. The socketChannel read and write can be optimized by manipulating chunks of data at a time rather than a single event at a time.

IX. Conclusion

In this project, we created a benchmarking setup for the Complex Event Processing Engine - Esper and evaluated parameters for a few queries. The results of the evaluated queries are as populated in the tables in the Results section.

Acknowledgment

I would like to thank Prof. Umesh Bellur for his guidance through the R&D Project. I would also like to thank YourKit project for the evaluation license for the profiler. Also, thanks to Sysads@CSE for allowing two VMs in the servers.

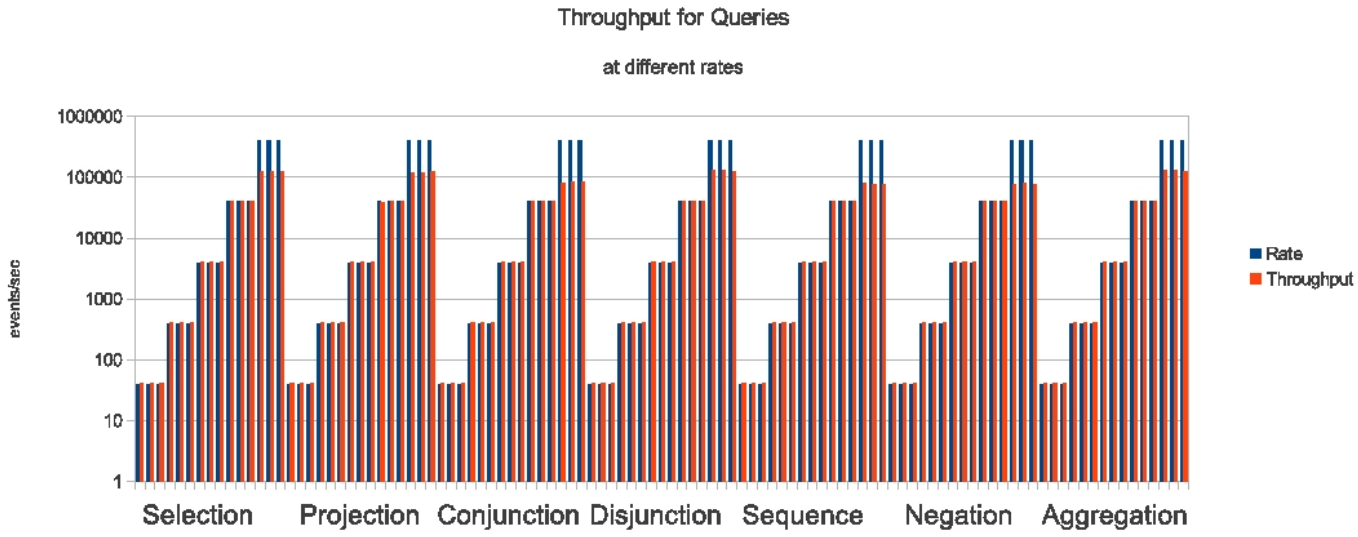

Fig. 4. Throughput with Rate

TABLE I. CLASSES OF QUERIES TO BE EVALUATED

Class of Query	**Query**	Selectivity
selection	select a from Market as a where a.test = 0 and a.volume < 5	7.40
projection	select price as a from Market where test = 0	16.67
conjunction	select a,b from pattern [every (a=Market and b=News)] where a.test = 0 and b.test = 0	13.25
disjunction	select * from pattern [every Market(ticker='S0AA',test=0) or every News(ticker='S1AA',test=0)]	3.33
sequence	select a,b from pattern [every (a=Market -> b=News(ticker=a.ticker))] where a.test = 0 and b.test = 0	10
negation	select a,b from pattern [every ((a=Market@consume -> b=News@consume) and not Market)] where a.test = 0 and b.test = 0	6.67
aggregation	select a, ticker, avg(price), count(*), sum(price) from Market(ticker='S0AA', test=0) as a	1.67

TABLE II. PERFORMANCE FIGURES

Query	**Avg Throughput (events/sec)**	**Avg Latency (ns)**	**Avg CPU (%)**	**Avg Mem (MB)**	**Match %**
Selection	1,25,834.00	3216.33	97.55	135.47	7.4
Projection	1,21,246.00	2706	97.25	137.32	16.67
Conjunction	83922.67	7114.67	98.19	133.9	13.33
Disjunction	1,28,842.33	3485.33	97.11	138.08	3.33
Sequence	77956.33	8337.67	98.01	136.8	10
Negation	77050.67	8904.67	98.47	135.69	6.67
Aggregation	1,28,120.00	5135.67	97.26	142.15	1.67

REFERENCES

[1] Esper Performance Benchmark page: http://docs.codehaus.org/display/ESPER/Esper+performance

[2] YourKit Profiler: http://www.yourkit.com/

[3] Esper CEP Ecosystem: http://blog.octo.com/en/the-esper-cep-ecosystem/

[4] http://www.codehouse.espertech.com/download/public/esperha.pdf

[5] ESP vs CEP : http://www.complexevents.com/2006/08/01/whats-the-difference-between-esp-and-cep/