Elastic Complex Event Processing under Varying Query Load

Thomas Heinze[1] Yuanzhen Ji[1] Yinying Pan[1] Franz Josef Grueneberger[1]
Zbigniew Jerzak[1] Christof Fetzer[2]

[1]SAP AG
Dresden, Germany
firstname.lastname@sap.com

[2]System Engineering Group, TU Dresden
Dresden, Germany
christof.fetzer@tu-dresden.de

ABSTRACT

Distributed data stream processing systems, like Twitter Storm or Yahoo! S4, have been primarily focusing on adapting to varying event rates. However, as these systems are becoming increasingly multi-tenant, adaptation to the varying query load is becoming an equally important problem.

In this paper we present FUGU – an elastic allocator for Complex Event Processing systems. FUGU uses bin packing to allocate continuous queries to a varying set of nodes. Driven by elasticity requirements FUGU maximizes the overall system utilization while trying to maintain stable processing latencies.

The specific contributions of this paper are: (1) introduction of a re-balancing scheme for bin packing allowing FUGU to increase overall system utilization by six percent and (2) a detailed study of achievable system utilization and latency under real-life workload from Frankfurt Stock Exchange.

1. INTRODUCTION

Distributed complex event processing (CEP) has been commonly used in context of financial trading systems [1]. Typical CEP use cases in financial domain usually revolved around single user, single query usage pattern. However, with recent proliferation of CEP in industries such as manufacturing [8] or analytics [11] the usage pattern is switching towards multiple users, multiple queries per system. The implication of this trend is the need for CEP systems to be able to accommodate not only varying event load but also varying query load.

In order to avoid constant overprovisioning and to be able to handle sudden load surges distributed CEP systems must be able to scale both in and out. Being able to scale both in and out while maintaining high overall system utilization is the ultimate goal of an elastic system [2]. Elasticity is an important property of every distributed system as it ensures its economic feasibility while being executed on any cloud platform.

Several authors have studied building elastically scalable complex event processing systems [7, 12]. However, we are not aware of a work which would explicitly target the problem of the varying query load in elastic CEP systems. In this paper we present the design and evaluation of the elastic allocation component – FUGU. FUGU can dynamically allocate and de-allocate both stateless and stateful queries in order to meet the utilization goals. To that end FUGU relies on bin packing to allocate queries to hosts.

The contributions of this paper are following: (1) we present a re-balancing extension of a state of the art bin packing approach [4], which allows to improve the average utilization of the system by up to 6% and (2) we present a detailed evaluation of the achievable utilization as a function of a given utilization target. The evaluation of our elastic allocation component has been performed on top of a commercial distributed complex event processing system using tick data streams from Frankfurt Stock Exchange.

2. SYSTEM ARCHITECTURE

Figure 1 shows the FUGU component and its interaction with the underlying CEP system. The underlying CEP system consists of several instances of a CEP engine running in parallel on heterogeneous hosts. The CEP system accepts and processes continuous queries consisting of direct acyclic graphs of operators. Our system supports primitive relational algebra operators (selection, projection, join, aggregation) as well as additional CEP specific operators (sequence, source and sink). Each operator can be executed on an arbitrary host. Therefore, the computation of a query can be partitioned over multiple hosts. The number of hosts is variable and dynamically adapted to the changing resource requirements by the FUGU component. FUGU is always provisioning one or two hot hosts to allow for a fast scale out [6].

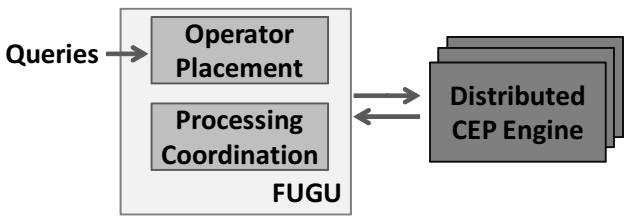


Figure 1: System architecture

FUGU is a centralized component. The role of FUGU is twofold: (1) it coordinates different instances of the CEP

engine and (2) it calculates placement decisions. When a new query is added or an existing query is removed, a bin packing algorithm is used (see Section 3) to calculate the operator to host assignment. When a new operator needs to be placed, FUGU will always try to locate a host with enough available resources to host this operator. If no such host can be found, a new host will be assigned to the system. When all operators on a certain host are removed, the host is then released by the system.

As soon as such an assignment has been derived, FUGU coordinates the placement of new and re-placement of existing operators. To that end FUGU communicates with all involved hosts using a topic-based publish/subscribe protocol. Newly added operators subscribe to their predecessor operators. Data published by an operator is sent to all subscribers. FUGU supports re-placement of both stateless (source, filters, projection, sinks) as well as stateful operators (aggregation, join, sequence) using a state transfer protocol similar to the one of [13].

3. OPERATOR PLACEMENT

The foundation of our operator placement approach is a load model, which estimates and measures CPU, memory and network consumption for each individual operator. When new queries are added, all variables in the model are first estimated using a worst case assumption. These values are subsequently updated during runtime with precise measurements.

The required CPU load ($load_{\text{CPU}}$) for a given operator (op) is calculated based on the operator's input rate ($input(op)$) and its per event processing time ($proc(op)$):

$$load_{\text{CPU}}(op) = proc(op) \cdot input(op) \tag{1}$$

During the estimation phase, we assume that the processing time of a new operator is comparable to the processing time of currently running/previously executed operators of the same type. The input rate is derived based on the input rate of the predecessor operators and estimations of their selectivities in a fashion similar to the approach presented by Viglas et al. [14]. For the purpose of the estimation we constantly measure the source input rate and use the maximum value observed so far. The major advantage of this scheme is that it only requires the input rates of the sources.

We use similar approach to estimate operators' memory and network consumption. The network bandwidth is derived from the operators' input and output rates, their selectivity (predicate) and the average size of input and output events. The memory consumption is estimated using a linear model which multiplies the operators' event rate by the window size and event size. The network consumption model is placement-aware: operators placed on the same host are assumed to communicate via in memory message passing. Operators on different hosts are assumed to communicate via network.

3.1 Elastic Operator Placement

The placement is calculated using a global bin packing algorithm [5] in a fashion similar to the one proposed by Backman et al. [4]. Bin packing algorithm calculates an assignment of items (operators) to bins (hosts) in a way that a minimal number of bins is used. The major criteria for assignment is the required CPU load of an operator. In addition, hosts with insufficient memory or network bandwidth are removed from the list of potential target hosts. FUGU uses a FirstFit bin packing variant, which assigns a newly added operator to the first node with enough remaining capacity.

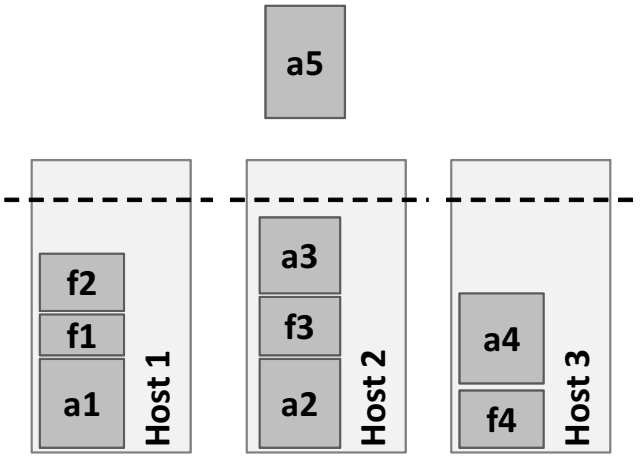


Figure 2: Example for applying the rebalancing heuristics

The placement algorithm can be configured to aim for a certain target utilization. This is realized by an additional user-defined parameter: the utilization threshold $thres$. The $thres$ value is used as the available capacity of a host, which should not be exceeded by the bin packing algorithm.

3.2 Re-balancing Heuristics

The above bin packing approach allows to scale out and to scale in with a changing number of queries. However, after evaluating this approach we have observed that the system is often reporting suboptimal utilization values – see Section 4. This is caused by the fact that remaining operators are scattered across all hosts in the system. This, in turn, prevents FUGU from releasing these hosts.

There exist two alternative approaches towards solving this issue. Either the bin packing algorithm is re-executed for all operators left in the system or specific operators are selected and re-placed so as to release least loaded hosts. A re-execution of the bin packing approach with all remaining operators would provide the best solution, however, it would also result in a large amount of operators and state being moved. This in turn would negatively impact the availability of the system. Therefore, in order to minimize the impact on the system availability we have implemented a re-balancing approach.

As soon as a query is removed, the re-balancing algorithm calculates the currently required minimal number of hosts ($host_{\min}$):

$$host_{\min} = \left\lceil \frac{\sum_{\forall op} load_{\text{CPU}}(op)}{thres} \right\rceil \tag{2}$$

In case the current number of hosts used by the system is larger than the calculated minimal number of hosts ($host_{\min}$) a re-balancing is triggered. During re-balancing only operators from hosts with the minimal load are subject to bin packing. Bin packing is executed for these operators until the total number of used hosts reaches $host_{\min}$.

An additional heuristic is used to detect imbalance during addition of queries. Let us consider the scenario shown in

Figure 2, where three active hosts are used and a new operator $a5$ should be placed. None of the hosts has enough remaining capacity to allow an assignment of the operator $a5$. Therefore, a new host needs to be allocated and the $a5$ operator needs to be placed on this host. However, if we consider the total remaining capacity on all hosts it should be possible to place the operator without allocating any new hosts. The re-balancing is triggered if during the addition of an operator op a new host should be allocated and the following condition holds:

$$load_{\text{CPU}}(op) < (n \cdot thres - \sum_{\forall o} load_{\text{CPU}}(o)) \qquad (3)$$

where n is the number of currently active hosts in the system and $\forall o$ represents all operators currently running in the system.

For re-balancing we choose the host, where the difference between remaining capacity and the newly assigned operator load is minimal. For this host we use a subset algorithm [10] to identify a minimal set of operators to redistribute in order to make place for the new operator to be added. We use the algorithm to calculate all valid solutions with a summed CPU load within the interval $[load_{\text{CPU}}(op), load_{\text{CPU}}(op) + int]$, where int describes the interval size. From this set we select the solution, which requires the smallest amount of state to be moved. Considering the example in Figure 2, Host 3 will be selected as one with the closest remaining capacity. Subsequently, operator $f4$ will be selected and moved to Host 1 and operator $a5$ will be placed on Host 3.

4. EVALUATION

We have implemented FUGU on top of a state of the art, commercial, distributed CEP engine. We have extended the underlying CEP system with capabilities required for dynamic host addition and removal as well as state migration. The evaluation is conducted in a shared, private cloud environment with up to 10 hosts with 2 cores and 4 GB RAM each. For evaluation we use a real-world tick stream from the Frankfurt Stock Exchange. We can replay the tick stream with a variable or a fixed data rate. For evaluating our system we use the following query template:

```
SELECT avg(price) FROM tickStream WITHIN x SEC
GROUP BY comp WHERE sector=y;
```

The above query calculates the average price for each company within a certain sector. The query workload is made variable by choosing the window size (x) and the sector (y) randomly. The query workload pattern was extracted from a web server log [3] – see Figure 3(a) and 3(b).

Performance is evaluated based on the end to end latency. We define the end to end latency as the difference between the time an event enters the system via source operator and the time it leaves the system via sink operator. Due to different complexities of queries the end to end latency of different queries can not be easily compared. Instead, for each query we calculate the ratio between the initial latency measured for the first ten seconds after the query has been added and the current end to end latency. We label this value as *latency ratio*. Latency ration should be ideally always equal to 1.

4.1 Elastic Scaling of FUGU

The goal of the first experiment is to demonstrate that FUGU is able to elastically scale the underlying CEP system

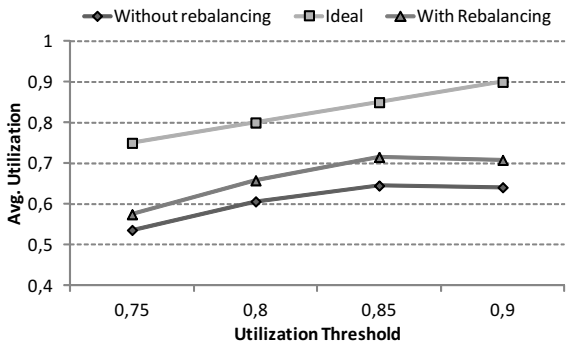


Figure 4: System utilization as a function of utilization threshold *thres*

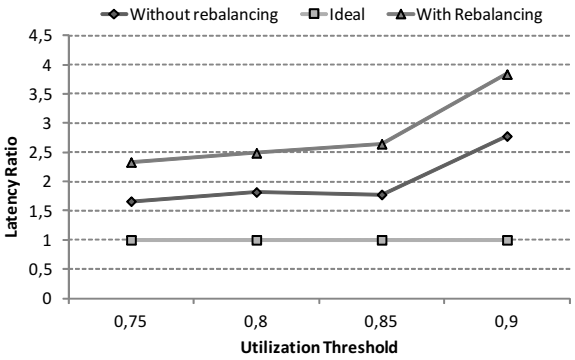


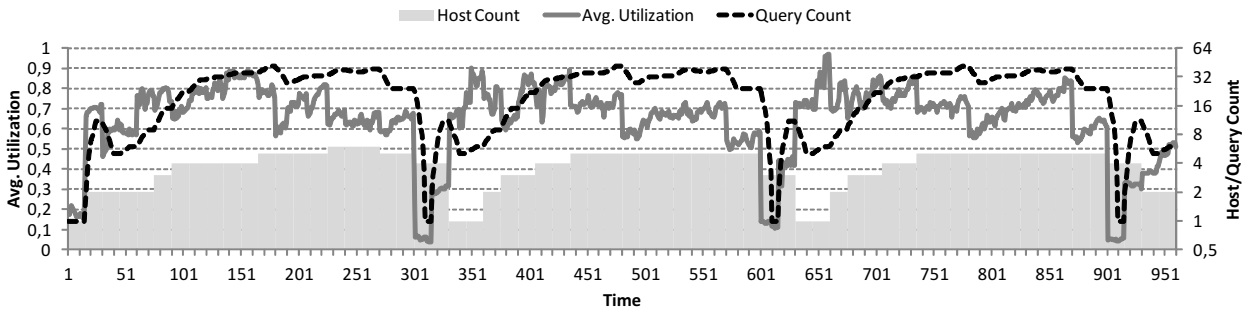
Figure 5: Latency ratio as a function of utilization threshold *thres*

with a varying number of queries. For this experiment we set the utilization threshold *thres* to 0.85. Figure 3(a) shows the average system utilization and used hosts count as a function of the query count. During peak load system runs 45 queries in parallel across six hosts. It can be observed that FUGU automatically scales underlying CEP system out and in depending on the query workload. The average utilization remains constant and oscillates around 60%. Figure 3(c) shows the corresponding maximum latency ratio across all queries running in the system. The average latency ratio of all queries stays close to 1, however certain queries experience short latency peaks. According to expectations this behavior manifests itself mainly during reconfigurations of the system, i.e., addition or removal of hosts.

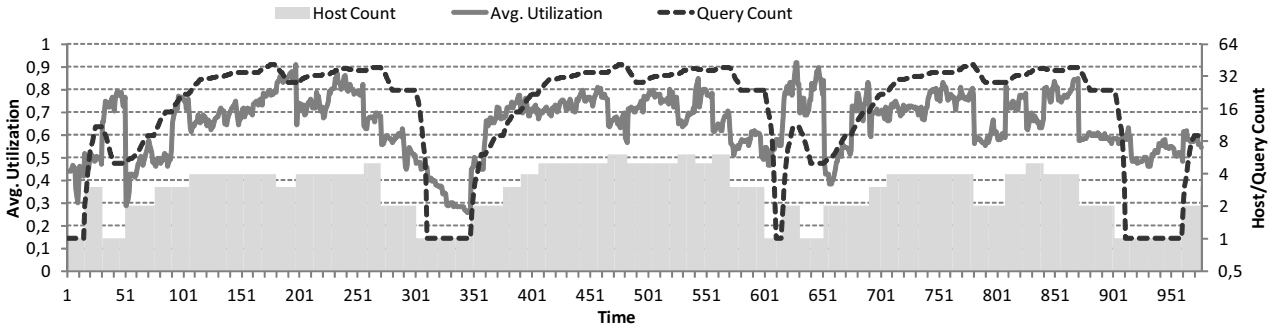
In the following experiment we have enabled the re-balancing algorithm and re-executed the experiment. Figure 3(b) shows that the system is able to release hosts earlier and in average uses less hosts than the approach without re-balancing. The average utilization increases to 65%. However, due to the re-balancing more peaks in the latency ratio can be observed – see Figure 3(d). This confirms the existence of a basic intuitive trade-off: the more aggressive the elasticity policy the less stable the system becomes.

4.2 Achievable Utilization

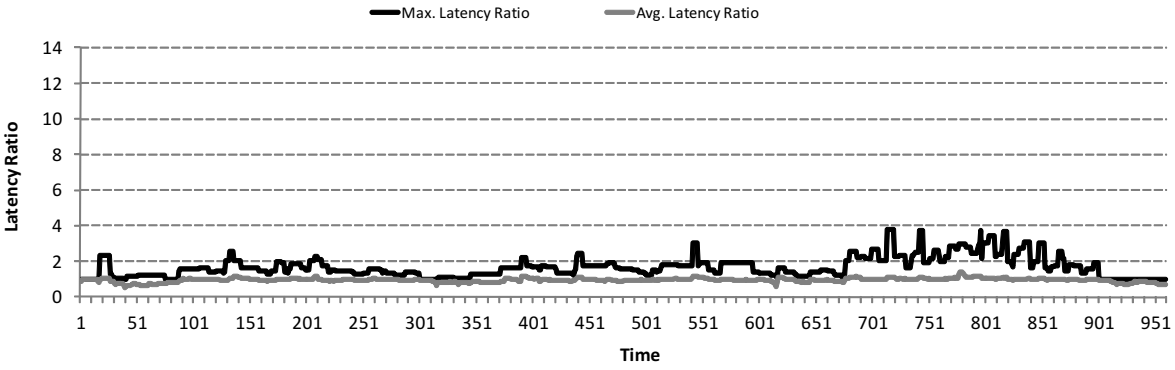
The goal of FUGU is to maximize the system utilization without significantly impacting the end to end latency of the running queries. In order to study the maximal achievable utilization of our system we changed the threshold *thres*



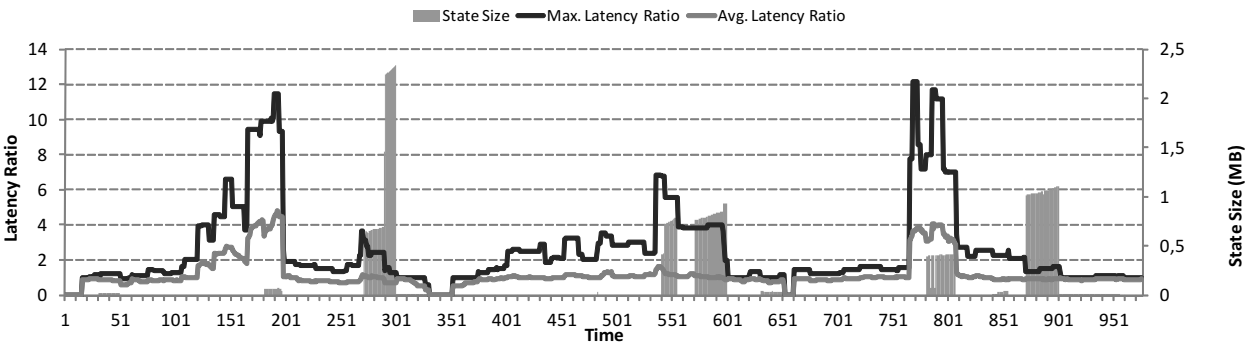
(a) Elastic scaling – average system utilization as a function of the query count



(b) Elastic scaling with re-balancing – average system utilization as a function of the query count



(c) Elastic scaling – maximum and average latency ratios



(d) Elastic scaling with re-balancing – maximum and average latency ratios and migrated state size

Figure 3: Elastic scaling with and without re-balancing

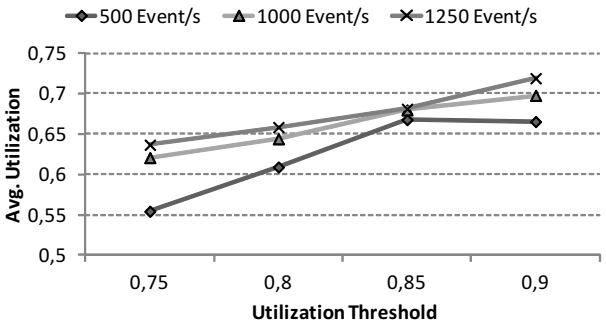


Figure 6: Average utilization as a function of varying event rate and utilization threshold

for the upper bound of utilization per host from 75% up to 90%. Figure 4 shows the resulting system utilization as a function of the threshold *thres*. In addition, Figure 5 shows the average latency ratio. We can observe that the achievable utilization increases from 53% for $thres = 0.75$ to 64% for $thres = 0.9$ while the latency ratio increases from 1.6 to 2.7. The maximal achievable utilization saturates starting from a value of utilization threshold $thres = 0.85$.

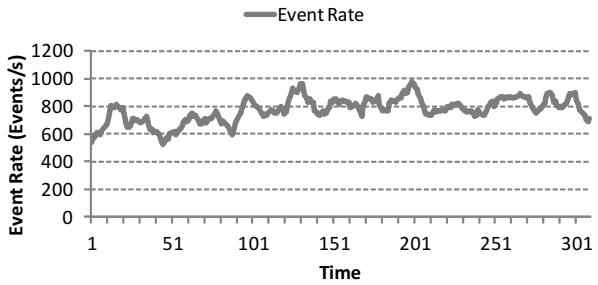
By using the re-balancing scheme the average utilization can be improved by up to six percent points, e.g. for $thres = 0.9$ to 70%. The maximal latency ratio increases to 3.8. The maximal latency ratio is proportional to the frequency with which the re-balancing is executed.

4.3 Influence of Event Rate

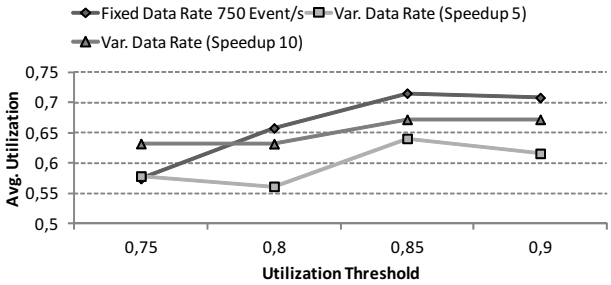
We have also measured the achievable utilization as a function of different stable event rates – see Figure 6. We have varied the event rate between 500 and 1250 events per second. The number of queries is identical as in case of the previous experiment. The number of hosts is automatically changing from 3 hosts for 500 events per second run with threshold 0.9 up to 9 hosts for 1250 events per second run with threshold of 0.75. From the experiment we can conclude that no linear correlation between the input rate and the achievable utilization can be drawn. This indicates that setting a good utilization threshold for different system conditions is a challenging problem.

To emphasize this result we re-ran above experiment with a varying event rate – see Figure 7. For this experiment we have fixed the utilization threshold at 0.9. The event rate pattern over time is shown in Figure 7(a). The event rate changes between 300 and 600 events per second for a speedup value of 5, and between 400 and 1000 events per second for a speedup value of 10.

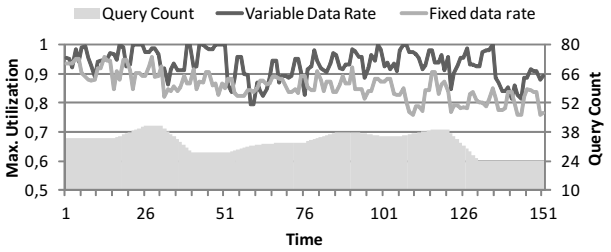
Figure 7(b) shows that the average system utilization, in case of variable event rate, is lower than in case of a fixed event rate. It is also, to a large extent, independent of the selected utilization threshold. Moreover, we have observed (see Figure 7(c)) that for individual hosts the utilization threshold is often exceeded. These two observations show a need to combine our approach with run-time adaptation and elasticity policies [6, 7], in order to be able to efficiently handle varying event rates.



(a) Variable event rate pattern for a speedup value of 10



(b) Average system utilization as a function of the utilization threshold and data rate



(c) Maximum utilization of an individual host

Figure 7: Variable event rate evaluation

4.4 Discussion

Based on the above evaluation we can conclude that our approach is well suited for elastic scaling with a varying number of queries. The system under control of FUGU is able to dynamically adjust the number of hosts and is able to keep the latency ratio close to 1 for the presented scenario. By trying to maximize the utilization we have also demonstrated that a trade off between latency ratio and achievable utilization exists. Specifically, finding a good upper threshold for the utilization of the system seems to be both important and non-trivial.

We have also outlined, that the event rate has a major influence on the achievable utilization. Especially, in case of varying event rates the system utilization significantly decreases. This requires the addition of run-time adaptation to FUGU, which we consider as future work.

5. RELATED WORK

Elasticity in context of data stream processing systems has been studied by various authors [6, 7, 12], however, none

of the proposed approaches considered a varying query load. Schneider et al. [12] present a scheme for elastic resource scaling within a single node. The system can adapt the number of threads used by a single operator to be able to handle varying event rate. Other approaches focus on adapting a distributed data stream processing system to changing event rates. Gulisano et al. [7] describe a distributed system using an upper and a lower bound on the load variance to trigger operator migration whenever these bounds are violated. The implication of this approach is the possibility of allocation of new hosts and thus worsening of the overall system utilization. Fernandez [6] et. al. present an integrated solution for dynamic scale-out and fault tolerance. Presented system supports check-pointing-based fault tolerance and policy-based scale out. However, it is not possible to scale the system in, therefore, unlike FUGU, it cannot be considered as fully elastic.

Balancing the load among hosts of a streaming system is related to a class of algorithms used for operator placement [15, 4]. Operator placement algorithms can target different objectives, most common being: end to end latency, network bandwidth and load (im-)balance – see [9] for a comprehensive survey of placement strategies. Xing et al. [15] presents an algorithm which balances the load between all hosts of the system by minimizing the load variation between hosts. FUGU uses similar technique where an initial assignment is optimized by partial re-balancing. However, the approach of Xing et al. only works for a fixed number of hosts, whereas FUGU can adjust the number of hosts dynamically. Backman et al. [4] present an approach, which balances the load between hosts using bin packing. Using simulation Backman et al. conclude that the system is able to provide latency guarantees. Evaluation with FUGU demonstrates that this claim is difficult to uphold in a system with a dynamic set of queries. Moreover, it is in opposition to the high utilization goal of elastic systems.

6. CONCLUSION

In this paper we have presented FUGU, an allocation component for distributed complex event processing systems. FUGU is able to elastically scale in and out the underlying CEP system with a varying query load. We have evaluated FUGU using real life workloads and demonstrated that it can achieve a good average utilization with a stable latency ratio. We have also presented a re-balancing extension allowing to migrate stateful and stateless operators between hosts, thus improving the overall system utilization by up to 6%.

For the future we plan to investigate how to improve the ratio between achievable utilization and measured latency. We also plan for provisioning QoS guarantees for a system under the control of FUGU. In addition, we want to extend the system to allow for run-time adaptation to dynamically changing event rates.

7. REFERENCES

[1] A. Adi, D. Botzer, G. Nechushtai, and G. Sharon. Complex event processing for financial services. In *SCW 2006: Proceedings of the 2006 IEEE Services Computing Workshops*, pages 7–12, 2006.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, April 2010.

[3] M. Arrington. AOL proudly releases massive amounts of private data. *TechCrunch: http://www. techcrunch. com/2006/08/06/aol-proudly-releasesmassive-amounts-of-user-search-data*, 2006.

[4] N. Backman, R. Fonseca, and U. Çetintemel. Managing parallelism for stream processing in the cloud. In *Proceedings of the 1st International Workshop on Hot Topics in Cloud Data Processing*, page 1. ACM, 2012.

[5] E. Coffman Jr, M. Garey, and D. Johnson. Approximation algorithms for bin packing: A survey. In *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co., 1996.

[6] R. C. Fernandez, M. Migliavacca, E. Kalyvianaki, and P. Pietzuch. Integrating scale out and fault tolerance in stream processing using operator state management. In *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*, 2013.

[7] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente, and P. Valduriez. StreamCloud: An Elastic and Scalable Data Streaming System. *IEEE Transactions on Parallel and Distributed Systems*, 23(12):2351–2365, 2012.

[8] Y. Ji, T. Heinze, and Z. Jerzak. HUGO: Real-Time Analysis of Component Interactions in High-Tech Manufacturing Equipment. In *DEBS 2013: Proc. Of the 7th ACM International Conference on Distributed Event-Based Systems*, 2013.

[9] G. T. Lakshmanan, Y. Li, and R. Strom. Placement strategies for internet-scale data stream systems. *Internet Computing, IEEE*, 12(6):50–60, 2008.

[10] S. Martello and P. Toth. Algorithms for knapsack problems. *Surveys in combinatorial optimization*, 31:213–258, 1987.

[11] C. Mutschler, H. Ziekow, and Z. Jerzak. The DEBS 2013 grand challenge. In *DEBS 2013: Proc. Of the 7th ACM International Conference on Distributed Event-Based Systems*, 2013.

[12] S. Schneider, H. Andrade, B. Gedik, A. Biem, and K.-L. Wu. Elastic scaling of data parallel operators in stream processing. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–12. IEEE, 2009.

[13] M. A. Shah, J. M. Hellerstein, S. Chandrasekaran, and M. J. Franklin. Flux: An adaptive partitioning operator for continuous query systems. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 25–36. IEEE, 2003.

[14] S. D. Viglas and J. F. Naughton. Rate-based query optimization for streaming information sources. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 37–48. ACM, 2002.

[15] Y. Xing, S. Zdonik, and J.-H. Hwang. Dynamic load distribution in the borealis stream processor. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 791–802. IEEE, 2005.