# Elastic Complex Event Processing

Thomas Heinze
SAP Research Dresden
Dresden, Germany
Thomas.Heinze@sap.com

## ABSTRACT

Complex Event Processing (CEP) systems are designed to process large amount of information by simultaneously evaluating multiple queries over event streams. Two main requirements imposed by the users of the CEP systems are: (1) ability to process high throughput event data and (2) ability to answer queries with very low latency. In order to meet above requirements CEP systems are becoming increasingly distributed. Distribution of queries as well as event streams across multiple nodes facilitates increasing the throughput of CEP systems while simultaneously maintaining low response times.

The widespread adoption of cloud computing and the accompanying pay-as-you-go model has added new dimensions to the problem of complex event processing in a distributed system. Nowadays, it is not only important to be able to scale the processing out to a large number of nodes, it is also equally important to be able to scale the processing down, as soon as the load or user requirements decrease. The ability to scale processing up *and* down along with the load and user requirements is called elasticity.

The goal of the thesis described in this paper is to develop a component allowing for elastic scaling of distributed CEP systems in response to variations in the load and contractual obligations regarding the quality of service. To this end, the thesis described in this paper will address following three major topics: (1) multi query optimization, (2) operator placement in distributed environments, and (3) cost efficiency. This paper outlines the state of art for the three aforementioned topics and presents the overall draft of the solution for the problem of the elastic complex event processing.

## 1. INTRODUCTION

Complex Event Processing (CEP) systems [21] are used in various scenarios, including (but not limited to): stock trading [20], click stream monitoring [11], and information integration workflows [7]. Moreover, Complex Event Processing is becoming a central component of Business Intelligence (BI) systems, as it facilitates solving BI task in (near) real time by short cutting the pipeline between raw data and the dashboard layer [7]. In contrast to classical database systems, CEP systems avoid persisting of incoming events, processing the data directly in memory. This allows CEP systems to process event streams with several thousands of events per second with sub millisecond latency [34].

With the increasing amount of data available for analysis [14] and increasing complexity and availability of the Business Intelligence analysis tools, single machine-based solutions are quickly becoming overloaded. Based on reports from SAP customers one can currently expect that CEP systems will have to cope with as much as 22.000 simultaneous users with estimated 5 queries being asked by every single user. The Options Price Reporting Authority (OPRA), which aggregates all quotes and trades from options exchanges, estimated peak rates of 456,000 events/sec (July 2007), with rates roughly doubling every year [33]. Since load shedding is not an applicable strategy when the correctness of results is required, distribution of the complex event processing emerges as the only viable option. First attempts at creating distributed CEP systems have been already undertaken and have resulted in systems, like: RAPIDE [22], Borealis [1], and DCEP [28].

However, in order to be able to competitively run large-scale distributed systems, cloud computing [8] resources have to be used. The key feature provided by cloud computing is the notion of elasticity. Elasticity has been defined [24] as: *[. . . ] capabilities [which] can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.*

Elasticity involves not only transparent scale up of the system, but also transparent scaling down if the current setup is too big with respect to the current load or requested quality of service. In contrast, a classical distributed CEP system [22, 1, 28] is designed to always support for the peak load. Therefore, it is periodically scaled up in case the input rate increases. However, in practice, the load imposed on a distributed system is not increasing linearly, but shows an unpredictable variability – see Figure 1. Using a static resource provisioning either not enough resources are available (underprovisioning) and requests are dropped, or too many resources are used (overprovisioning), creating unnecessary costs. In contrast, using elastic resource provisioning, the system can alter its resource utilization dynamically, which


*MDS'2011,* December 12th, 2011, Lisbon, Portugal.

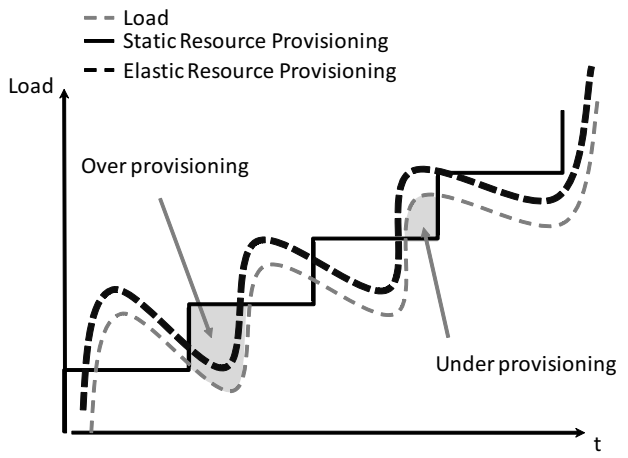


**Figure 1: Comparison of static and elastic resource provisioning – source: [35]**

avoids over- as well as underprovisioning.

Recently a number of prototypes for cloud-based CEP engines has been proposed, including: SEEP [25], an extension for System S [27], Flood [4], StreamCloud [12] and Kleiminger et al. [17]. In the following table (Table 1) we characterize aforementioned systems based on the level at which the elasticity is realized.

| Elastic | Action Taken | System |
|---|---|---|
| Engine | New engine instance is started and new queries are employed on this engine | [17] |
| Query | New query instance is started and the input data is split | [12, 25] |
| Operator | New operator instance is created and the input data is split | [4, 27] |

**Table 1: Different levels of elasticity and corresponding actions**

A CEP engine can be made elastic by parallelizing either at the engine, query or operator level. Each of these approaches has benefits and shortcomings. The engine level approach requires the smallest set of changes to the overall system, however it also introduces the largest overhead to initialize a new instance and exposes the smallest flexibility and granularity. The query level approach imposes less overhead, however it requires the system to be able to optimize for a possibly very large amount of queries – see section 4 for more details on this approach. The query level approach is also less flexible than the operator level approach. The operator level approach helps to scale the system in the most fine grained fashion. However, it also requires fully elastic operators, which are difficult to implement, especially if a system permits user defined operators. Moreover, too high degree of distribution might result in communication overhead becoming a limiting factor with respect to scalability [12].

In order to exploit elasticity a complex event processing system has to be designed in an elastic way, allowing for operation with a variable number of nodes. To be able to realize such systems, a solution for the following three issues has to be created [35]:

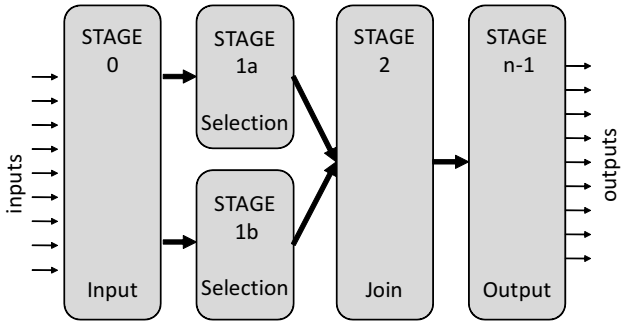


**Figure 2: Example dataflow on StreamMine**

1. Which components or layers in my application architecture can become elastic?
2. What will it take to make them elastic?
3. What will be the impact of realizing elasticity onto my overall system architecture?

The current focus of most CEP systems is on implementing answers to questions (1) and (2). None of the works has done an intensive study on the extensions needed for making the elasticity transparent to the user. The thesis described in this paper tries to fill this gap by researching question (3). The system needs to decide automatically, when to scale up and also when to scale down again to reduce the monetary cost. In addition, the system has to detect operators becoming bottlenecks and react by e.g. moving them to another node.

The remainder of this paper is structured as follows: Section 2 introduces the system model and general architecture of the CEP system which will be used as a foundation for the work carried out within the described thesis. In Section 3 main components of the proposed system are introduced, including multi query optimization, operator placement, and cost efficiency. Section 7 describes the planned evaluation strategy and plan for the whole system. Section 8 concludes this paper.

## 2. SYSTEM MODEL

The thesis described in this paper is part of the SRT-15[1] research project. The SRT-15 research project is funded by the European Commission within the Seventh Framework Programme (FP7) under the Grant Agreement number 257843 in the area of Internet of Services, Software & Virtualisation (ICT-2009.1.2).

A software foundation for both the SRT-15 research project and the described thesis the StreamMine [23] system. This StreamMine system is a framework for processes of events using a stage-based approach, where each stage can be programmed to express an independent logic unit and store independent state. Within a single stage multiple instances of the stage logic realize the same functionality for different data. In order to achieve this, StreamMine offers functionality to split the input data using a key-based approach – see Figure 2 for the architectural overview.

[1]Project homepage: `http://www.srt-15.eu`

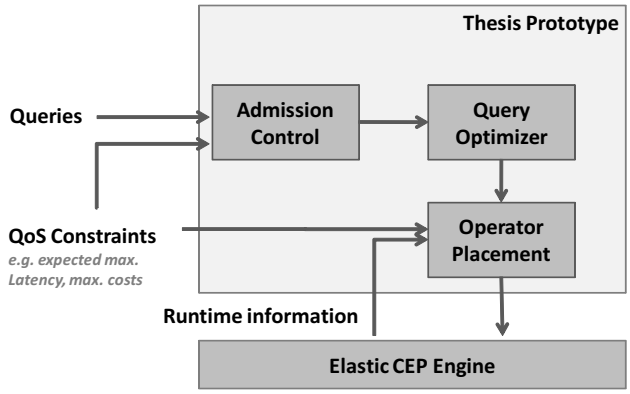


**Figure 3: Thesis prototype**

StreamMine can be executed in a distributed environment. For each execution all involved stages have to be assigned to physical underlying system nodes. Subsequently, stages have to be explicitly connected using point to point connections. This allows to scale the different stages based on expected event load by: (1) adding new stages to the already existing system or (2) adding additional nodes to already running stages. StreamMine can therefore be used to realize elasticity on an operator level.

Within the thesis, described in this paper, we define typical CEP operators (Selection, Join, Aggregation and Sequence) on top of the StreamMine model. We assume that input for the CEP system, running on top of StreamMine, consists of a set of infinite event streams and a set of continuous queries. The number of queries can change during runtime, as new queries can be added and removed dynamically. We also assume that queries submitted to the CEP system model the data flow as an acyclic graph. Figure 2 illustrates such data flow consisting of two Selection operators and one Join operator.

## 3. SYSTEM ARCHITECTURE

The main challenge (and simultaneously advantage) when designing a cloud-based CEP engine is that it must handle a rapidly changing set of events and queries, where both the number of queries and the event rate is very hard to predict.

As outlined in Section 1 peak load rates can be as high as 100,000 concurrent queries and 400,000 events per second. However, the normal load is expected to significantly differ from the above values, depending mostly on the application type. This in turn, requires the system to be able to automatically scale up and down along with the varying load, no user interaction is involved in the elastic scaling process. In order to allow for such functionality we propose a system architecture presented in Figure 3.

The key concept of the thesis prototype with respect to its architecture, is that it will be decoupled from the underlying elastic CEP system. All communication between the thesis prototype and the underlying CEP system will be executed via generic interfaces. This allows to use of the thesis prototype in combination with any elastic CEP engine, as long as the underlying CEP engine implements the specified interfaces and the required functionality including a distributed execution, dynamic addition of queries, operators and operator instances as well as dynamic movement of operators instances to other nodes. The prototype consists of three major components:

**Multiple query optimization** Given a large amount of concurrently running queries it is likely that results produced by parts of or even whole queries are identical. In order to lower the load on the system such overlaps should be exploited. A solution can be based on multiple query optimization [29] – a technique which allows creation of global query plans reusing common results.

**Operator placement** Based on a global query plan and a set of available nodes an assignment of operators or queries onto the physical nodes has to be performed. In order to accommodate for the varying load operator placement has to be aware of the current load levels as well as the current utilization of system nodes. Moreover, considering a deployment in the cloud, the operator placement component must be able to request and release resources as needed in order to cope with the varying workload and quality of service requirements like a maximal end to end latency or an expected minimal throughput.

**Cost efficiency** In most scenarios users provide not only event sources and queries to run against them but also a monetary budget and quality of service requirements which need to be met. Therefore, both aforementioned components must be able to operate in a cost efficient way. This means that the query plan and its execution must be tuned in such a way as to minimize the execution cost, while simultaneously meeting the specified quality of service requirements. In addition, newly added queries should be rejected by an admission control in case they can not be executed with the given budget.

In the following sections we will describe each of the aforementioned components in more detail. Specifically, we will motivate the design decisions regarding the components' design and we will present related work and the progress beyond the state of the art.

## 4. MULTIPLE QUERY OPTIMIZATION FOR ELASTIC CEP

The goal of the Multiple Query Optimization [29] (MQO) is to reduce the number of concurrently running queries in a CEP and/or database system. A MQO approach involves identification and reuse of identical results produced by operators within a global query plan. MQO allows avoiding repetitive computation of identical results and is specifically well suited for CEP systems, which are working with long running queries [7]. Figure 4 illustrates the multi query optimization approach.

The solution to the MQO is a global query plan incorporating multiple queries. The goal of the MQO when creating the global query plan is to minimize a cost metric, such as the sum of all operators cost. The task is known to be NP-complete [30], because the global optimal plan cannot be constructed by simply combining optimal per query plans. Instead, all possible query plans for all queries have to be examined, which leads to an exponential growth of the

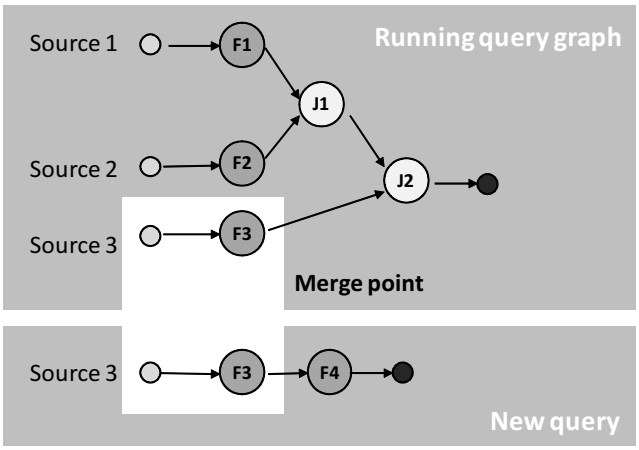


**Figure 4: Example for an incremental multi query optimization**

search space. To avoid this problem most approaches use heuristics [26] to find a near optimal query plan.

The MQO approach used in context of this thesis should be incremental, allowing adding new queries and removing queries during runtime. The global query plan should only contain the current active queries to allow an efficient resource usage. Many classical approaches used in database systems [6, 26] are not incremental algorithms, which means that each time a new query arrives the computation to find the optimal query plan has to be restarted. This can become very expensive, especially when the query arrival frequency is high. In addition, when adding new queries the already running ones should not be interrupted, i.e., changes to the currently executed global query plan should be avoided. If the position of operators within a running global query plan is changed this might involve changes in the semantics of the current operator state (e.g. the previously seen and still valid events inside a join operator), which either results in a complex reinitialization or incorrect results.

In the context of data streaming the usage of multi query optimization was first addressed in [13, 16]. Both approaches show an efficient way to optimize a large amount of continuous queries. However, both approaches do not allow for addition nor removal of queries during runtime. A system for incremental query optimization has been first proposed in [15] – it uses internal index structures to easily identify possible merge points for new queries. Additionally, the authors use query subsumption [3] for filter expressions, which results in a higher results reuse. However, the set of operators supported in the above system is very limited and the operators are only executed on the same input stream.

The thesis described in this paper will extend the above approach to: (1) accommodate a full set of CEP operators as well as (2) multiple input streams and (3) query removal from the system. In addition, instead of comparing the semantics of the operators, a stream annotation approach will be used. In our approach every stream is annotated with a set of predicates based on the operator semantics defining tuples contained within this stream. The goal is to avoid query rewriting and make use of efficient index structures to allow for fast integration of new queries.

# 5. OPERATOR PLACEMENT FOR ELASTIC CEP

Given a query plan and a set of available nodes, it has to be decided which operators are to be mapped onto which nodes and how many nodes should execute a given operator. This problem can be solved by operator placement algorithms [19] which try to place a set of operators on a fixed set of distributed nodes. A number of approaches [2, 16, 32, 36] which try to optimize operator placement based on a specified metric have been proposed. Authors in [19] outline different design dimensions for operator placement algorithms, in the following we mention those which are relevant to the thesis described in this paper:

**Architecture** Existing approaches can be categorized being an independent module [16, 32] or as distributed logic [2, 36]. With respect to the architecture the thesis prototype described in this paper is implemented as a centralized and independent module.

**Metric** The operator placement can be optimized based on various metrics including load, latency, network bandwidth or operator importance. Within the thesis described in this paper different metrics including monetary cost (see Section 6) will be evaluated.

**Adjustment** The operator placement can either be static, with new placement calculated only when a new query arrives or departs, or dynamic with the placement being adjusted, subject to varying load and node utilization. Within the thesis described in this paper a dynamic operator placement based on the system load and utilization will be used.

With the thesis described in this paper two additional problems will be investigated: (1) it has to be decided, how many nodes per operator are needed and (2) how to perform an a priori (instead of an a posteriori) node provisioning.

The first problem can be solved in a two phased approach. First, an initial estimate based on the number of available nodes and the complexity of the operator is performed. Subsequently, after deployment has taken place, runtime information is used in order to further refine the number of nodes used to handle the operator load.

The second problem stems from the fact that the number of available nodes is not fixed. Nodes are purchased on demand from the cloud provider. Ideally, one should try to avoid situations where nodes are becoming bottlenecks (underprovisioning) as well as overprovisioning of the overall system. The major challenge is to be able to detect such situations early enough to migrate the operator to another machine before violating given QoS constraints. This is motivated by the fact that both allocation of a new node and moving of an operator to a newly allocated node are not instantaneous. Moreover, a frequent allocation and deallocation of nodes (called thrashing) as a result of a bursty load should be avoided.

The overall idea for the operator placement is to provide a fast and simple initial placement strategy and perform subsequent adjustments according to the varying load. The motivation is based on following facts: (1) in CEP systems, unlike in classical data bases, it is not possible to predict the load and (2) initial operator placement is only the starting point for the search of an (sub) optimal solution.

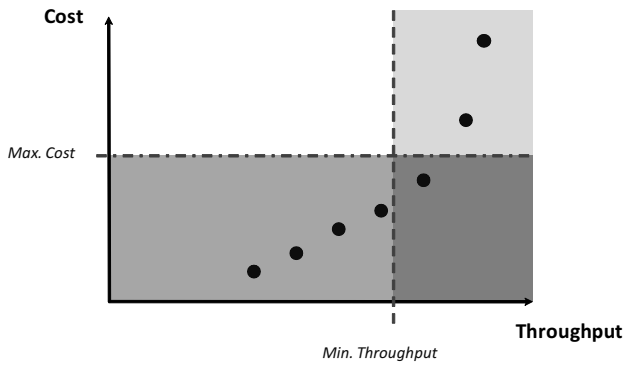


**Figure 5: Cost efficient execution of queries**

## 6. COST EFFICIENT ELASTIC CEP

Most cloud providers use a pay-as-you-go model, which means, that a user is charged for every CPU cycle he uses. The Elastic CEP prototype will use this property to reduce the information processing cost for the user as much as possible. Users of the Elastic CEP prototype will be able to specify a maximum cost for information processing along with QoS constraints, such as: a lower bound for the throughput and upper bound for the latency. The Elastic CEP prototype will try to optimize the data flow in such a way so as to stay within these constraints. From the large number of possible global query plans ones which fulfill both expected QoS constraints as well as the cost limits will be selected.

Figure 5 illustrates a set of possible query plans with expected throughput and associated cost along with an expected minimal throughput and a maximum budget. The chosen query plan should meets both criteria – which is true for the query plan in the dark gray area.

A number of authors [9, 18, 31] have studied the problem of cost efficient computations, especially in context of MapReduce [10] and cloud computing. For batch-based tasks it has been shown [18] how to integrate monetary costs into the schedule computation using an additional cost dimension during optimization. An approach proposed by authors in [9] shows how to combine a pricing model with a provisioning infrastructure and how to incorporate existing SLAs defined between customers and the service provider. The authors also demonstrate how service providers have to pay compensation fees in case of SLA violations. Authors in [31] present a heuristic method of optimization which speeds up a given task within a predefined budget by optimizing the number of allocated machines.

However, the complexity of cost efficient planning for applications involving streaming data is much higher, because the concrete rate for the event sources cannot be predicted. If the number of machines has to be increased (due to an increased amount of incoming events) the remaining budget has to be taken into consideration. As a result, the system has to decide (using an admission control) whether to reject a new set of queries, due to violation of the available budget.

## 7. EVALUATION

In order to be able to evaluate the Elastic CEP prototype a benchmark, containing both synthetic as well as real-world data, is needed. The goal of the evaluation is to show strengths and limitations of the Elastic CEP prototype with respect to handling varying load (event rates and number of queries) and the responsiveness of the system. The evaluation will examine following metrics:

**Throughput** The elastic CEP prototype must be able to demonstrate handling of query loads exceeding capacity of a single node. The Elastic CEP prototype should demonstrate the ability to execute a distributed global query plan.

**Query load** This metric will evaluate the multiple query optimization component – both the performance of an optimized query plan as well as the performance of the optimizer will be measured.

**Responsiveness** The responsiveness of the Elastic CEP prototype describes the time needed to adapt to changing query loads as well as the ability to detect overloaded operators.

**Cost efficiency** To test the cost efficiency the Elastic CEP prototype will be faced with situations in which queries have to be rejected or the available budget is limited. The system should show its ability to meet the budget constraints.

The evaluation will be based on real-world and synthetic test data. Therefore, existing benchmarks will be considered as starting point for the evaluation. The most common benchmark for CEP systems is Linear Road [5]. The Linear Roads benchmark simulates a toll system on a large number of highways. The system uses a fixed set of queries with well defined constraints regarding correctness and response times. The Linear Road benchmark comes with an event generator, which can change the amount of events sent per second. The main goal of Linear Road benchmark is to test the maximum event rate a system can sustain.

However, the both event rates and the query load show a linear behavior, which does not allow to test for the elasticity of the overall system. Elasticity has to be tested by additional test cases – therefore, one of the tasks of the thesis described within this paper is to develop a benchmark suite which can be used for comparing the performance of the elastic CEP systems.

## 8. CONCLUSION

The thesis described in this paper introduces an extension to existing CEP systems which reflects the high dynamism of a cloud environment. This involves studying the problem of: (1) how to handle a large amount of concurrent queries, (2) how to make use of allocated resources efficiently and (3) how to realize a cost efficient computation. Therefore, the Elastic CEP prototype developed within the thesis will accept following input: (1) a set of queries, (2) a set of event sources and a (3) set of available nodes and their utilization. As an output the component will provide an assignment of operators to nodes for an elastic CEP system taking into account the runtime statistics. This research is based on existing approaches for distributed complex event processing, however progress beyond the state of the art is made in that the dynamism of the cloud environment and load are considered. The Elastic CEP prototype will be evaluated using a new benchmark system allowing the comparison to other systems with respect to the support for elasticity.

# 9. REFERENCES


[1] D. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, et al. The design of the Borealis stream processing engine. In *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005), Asilomar, CA*, pages 277–289. Citeseer, 2005.

[2] Y. Ahmad and U. Çetintemel. Network-aware query processing for stream-based applications. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 456–467. VLDB Endowment, 2004.

[3] M. Al-Qasem and S. Deen. Query subsumption. *Flexible Query Answering Systems*, pages 29–42, 1998.

[4] D. Alves, P. Bizarro, and P. Marques. Flood: Elastic streaming MapReduce. In *DEBS'10*, pages 113–114, 2010.

[5] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear road: A stream data management benchmark. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 480–491. VLDB Endowment, 2004.

[6] S. Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 34–43, Seattle, Washington, June 1998. ACM Press.

[7] S. Chaudhuri, U. Dayal, and V. Narasayya. An overview of business intelligence technology. *Communications of the ACM*, 54(8):88–98, August 2011.

[8] M. Creeger. Cloud computing: An overview. *Queue*, 7(5):3–4, June 2009.

[9] I. Cunha, J. Almeida, V. Almeida, and M. Santos. Self-adaptive capacity management for multi-tier virtualized environments. In *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on*, pages 129–138. IEEE, 2007.

[10] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[11] M. Gualtieri and J. R. Rymer. The Forrester Wave: Complex Event Processing (CEP) Platforms, Q3 2009. Online, August 2009.

[12] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, and P. Valduriez. StreamCloud: A large scale data streaming system. In *2010 International Conference on Distributed Computing Systems*, pages 126–137. IEEE, 2010.

[13] M. Hong, M. Riedewald, C. Koch, J. Gehrke, and A. Demers. Rule-based multi-query optimization. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 120–131. ACM, 2009.

[14] A. Jacobs. The pathologies of big data. *Queue*, 7(6):10–19, 2009.

[15] C. Jin and J. Carbonell. Predicate indexing for incremental multi-query optimization. *Foundations of Intelligent Systems*, pages 339–350, 2008.

[16] E. Kalyvianaki, W. Wiesemann, Q. Vu, D. Kuhn, and P. Pietzuch. SQPR: Stream query planning with reuse. *Imperial College London, Tech. Rep*, 2010.

[17] W. Kleiminger, E. Kalyvianaki, and P. Pietzuch. Balancing load in stream processing with the cloud. In *Proceedings of the 6th International Workshop on Self Managing Database Systems (SMDB 2011), Hannover, Germany*. IEEE, 2011.

[18] H. Kllapi, E. Sitaridi, M. Tsangaris, and Y. Ioannidis. Schedule optimization for data processing flows on the cloud. In *Proceedings of the 2011 international conference on Management of data*, pages 289–300. ACM, 2011.

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

[20] N. Leavitt. Complex-event processing poised for growth. *Computer*, 42(4):17–20, 2009.

[21] D. Luckham. *The power of events: An introduction to complex event processing in distributed enterprise systems*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[22] D. Luckham and B. Frasca. Complex event processing in distributed systems. *Computer Systems Laboratory Technical Report CSL-TR-98-754. Stanford University, Stanford*, 1998.

[23] A. Martin, T. Knauth, S. Creutz, D. B. de Brum, S. Weigert, A. Brito, and C. Fetzer. Low-overhead fault tolerance for high-throughput data processing systems. In *ICDCS '11: Proceedings of the 2011 31st IEEE International Conference on Distributed Computing Systems*, pages 689–699, Los Alamitos, CA, USA, June 2011. IEEE Computer Society.

[24] P. Mell and T. Grance. The NIST definition of cloud computing. *National Institute of Standards and Technology*, 53(6), 2009.

[25] M. Migliavacca, D. Eyers, J. Bacon, Y. Papagiannis, B. Shand, and P. Pietzuch. SEEP: Scalable and elastic event processing. In *Middleware'10 Posters and Demos Track*, page 4. ACM, 2010.

[26] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *ACM SIGMOD Record*, volume 29, pages 249–260. ACM, 2000.

[27] S. Schneider, H. Andrade, B. Gedik, A. Biem, and K. 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.

[28] N. Schultz-Moller, M. Migliavacca, and P. Pietzuch. Distributed complex event processing with query rewriting. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, pages 1–12. ACM, 2009. operator placement.

[29] T. Sellis. Multiple-query optimization. *ACM Transactions on Database Systems (TODS)*, 13(1):23–52, 1988.

[30] T. Sellis and S. Ghosh. On the multiple-query optimization problem. *Knowledge and Data Engineering, IEEE Transactions on*, 2(2):262–266, 1990.

[31] J. Silva, L. Veiga, and P. Ferreira. Heuristic for resources allocation on utility computing infrastructures. In *Proceedings of the 6th international workshop on Middleware for grid computing*, page 9. ACM, 2008.

[32] U. Srivastava, K. Munagala, and J. Widom. Operator placement for in-network stream query processing. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 250–258. ACM, 2005.

[33] StreamBase. Real-time data processing with a stream processing engine. Technical report, 2008.

[34] Sybase. Evaluating Sybase CEP performance. Technical report, October 2010.

[35] J. Varia. Architecting for the cloud: Best practices. Technical report, Amazon, Inc., 2010.

[36] Y. Xing, S. Zdonik, and J. 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.