Multi-site Cooperative Data Stream Analysis

Fred Douglis* , Michael Branson, Kirsten Hildrum, Bin Rong† , Fan Ye
IBM

ABSTRACT

System S is a large-scale distributed streaming data analysis environment. Ultimately, we envision that there will be multiple sites running the System S software, each with their own administration and goals. However, cooperation between these sites can frequently be of mutual benefit. We are designing the framework to support numerous sites that can work both independently and in cooperative fashions, with a variety of interaction models such as peer-to-peer or federated. Depending on the degree of site autonomy and the relationships between any given pair of sites, the sites would be able to share data, perform processing on behalf of one another, or even take over tasks of a failed site. Interoperability is complicated by the degree of site autonomy as well as differences in execution environments and security policies. This paper surveys System S, describes its models for site interoperability, and discusses particular design issues such as site failover and heterogeneity.

1. INTRODUCTION

Data Stream Management Systems take continuous streams of input data, process that data in certain ways, and produce ongoing results. While a common way to structure these systems is to perform traditional database operations on the input streams [1, 7, 19], it is also possible to apply more general application logic, at the cost of higher complexity [18].

System S [3] is a project within IBM Research to enable very sophisticated stream processing. Its goal is to extract important information from voluminous amounts of unstructured and *mostly* irrelevant data. Example applications of such a system include financial markets (predicting stock value based on real-world events), responding to disasters such as Hurricane Katrina, or processing sensor data such as volcanic activity or telemetry from radio telescopes.

System S has a number of challenges, including:

Rapid reconfiguration The system must be able to adjust quickly to the changing requirements and priorities of its users and administrators. As it does so, It must simultaneously identify and incorporate new inputs into its processing and cope with the loss of existing data sources or processing capacity.

Expecting overload in the common case System S has been designed from the outset with the goal of functioning well under high load: the system is assumed to be in a constant state of overload and must continually adjust its resource allocations to support the highest priority activities. Applications are to be designed with a great deal of resilience to variations in processing resources, missing data (i.e., data that got pushed out by more important data [8]), available input streams, and so on.

Heterogeneity We expect there will be many System S *sites*, many of which have substantial capacity (thousands of processing nodes and terabytes to petabytes of storage), and some of which have more limited resources and provide specialized tasks such as data acquisition. Some may be operated by the same organization, such as IBM, but in general each site can be completely autonomous and have significant variation in its execution environment, policies, and goals. These autonomous systems may cooperate to varying extents and in very different ways, depending on the compatibility of any given set of sites.

The focus of this paper is the cooperation among multiple System S sites. The cooperation takes several forms:

- Passing "primal" streams, which are brought into one site from the outside, on to another site that needs to analyze the same input data.
- Passing "derived" streams, which are created by analysis of other streams in one site, on to another site.
- Providing execution resources to another site for handling overload. This overload may be due to sudden increases in the system workload or sudden decreases in available resources (due to partial failure).
- Shifting important processing to another site in the case of the total failure of a site.

All these interactions must be supported in the face of other requirements and challenges, such as site autonomy, privacy and security constraints, and differences in execution environments.

While significant pieces of a single System S site have been prototyped, evaluated, and reported elsewhere [3, 13], this

*Contact address: fdouglis@us.ibm.com
†Work done during an internship with IBM.

paper describes the early stages of an ambitious design for multisite operation. The rest of the paper is organized as follows. The next section describes System S in greater detail. Section 3 discusses the ways in which sites may interoperate. Section 4 describes *Common Interest Policies* (CIPs), the mechanism for dictating how multiple sites may interact. The next few sections cover specific aspects of multisite operation in greater detail: distributed data acquisition (Section 5), failover (Section 6), and heterogeneity (Section 7). The paper finishes with related work and conclusions.

2. System S

This section summarizes the architecture of System S and describes some of its key components:

Inquiry Services (INQ) Users pose *inquiries* to the system to answer certain high-level queries: for example, "Show me where all bottled water is in the hurricane area." A *planner* subcomponent [15, 16] determines appropriate compositions of data sources and processing in the form of jobs that can achieve the goals. It then submits such jobs to the Job Management component. The planner needs to take into account various constraints such as available input sources, the priority of the inquiry, processing available to this inquiry relative to everything else being produced by the system, privacy and security constraints, and other factors.

Data Acquisition There are many possible streams that an application can process, both primal streams from outside System S and derived streams created by PEs. A component called the *Data Source Manager* (DSM) matches applications with appropriate streams, based on the constraints (particularly data types and source quality) specified by the applications. It returns "data source records" that provide information to access these data sources; see Section 5 for details.

Job Management A job in System S is a set of interconnected *Processing Elements* (PEs), which process incoming stream objects to produce outgoing stream objects that are routed to the appropriate PE or to storage. The PEs can be either stateless transformers or much more complicated stateful applications. By combining these PEs, along with stored data, System S enables sophisticated stream mining applications. The PEs can run on the same or different processing nodes. The job manager within a site is responsible for initiating and terminating jobs. It works with an optimizing scheduler that allocates nodes to PEs based on their priority, inter-node connectivity, bandwidth requirements, and other factors.

Data Fabric At the lowest level of the system, the Data Fabric provides the transport of streams between PEs and into persistent storage. The storage system uses *value-based retention* [8] to automatically reclaim storage by deleting the least valuable data at any given time.

Each System S site runs an instance of each of these system components, possibly as a distributed and fault-tolerant service.

Each site may belong to a distinct organization and has its own administrative domain; i.e., administrators who manage one site generally have no control over another site. In this respect, distributing System S among multiple sites is similar to Grid Computing [9]: sites negotiate some sort of peering relationship, offering resources to each other, but retaining a great deal of local autonomy. As with the Grid, sites that want to collaborate for common goals and benefits can negotiate and form "virtual organizations" (VOs). The following sections discuss the ways in which System S sites can potentially interoperate and the method for specifying allowable interactions.

Finally, we note that prototypes of key elements of System S were completed and demonstrated over the past several months. There is currently a large effort to develop a single, integrated prototype and to design the extensions necessary to support the interoperation of multiple autonomous sites, the latter being the focus of this paper.

3. DISTRIBUTION MODELS

A single System S site provides a very powerful solution, capable of solving complex analysis problems, but cooperation between multiple sites enhances this capability in a number of areas:

Breadth of analysis An organization may utilize System S to address a set of problems that require analysis, processing all relevant data that it is able to access. Other organizations may do similar things. But the prospect of cooperation between two specialized organizations can create a much more diverse set of information to analyze, expanding the set of problems that could be solved, improving the quality of the output of the analysis, or offering additional types of analysis not available in a single organization. For example, a multinational financial services company might perform detailed acquisition and analysis of companies, economies, and political situations within the local geographic region of each of its analysis sites; these could interoperate minimally by default, but cooperate closely upon a significant event or when analysis of multinational organizations is required.

Reliability The reliability of one site can be significantly improved if it has agreements in place with other sites to take over key processing and storage upon failures.

Scalability Extreme scalability cannot be achieved through unbounded growth of an individual system. The cooperation of multiple autonomous systems can achieve much higher levels of scalability.

There are a number of possible ways that sites may interoperate. These are presented below as a spectrum of distribution models, ranging from the most basic approach to the most sophisticated. System S intends to support all of these distribution models in one form or another.

It is important to note that sites can be arranged to support a number of different peering models. The CIPs (see Section 4) define the relationships between sites, forming

virtual organizations. Different VOs may overlap with one another, resulting in a given site participating in multiple VOs. This allows for basic point-to-point (site-to-site) peering, as well as peering between entire VOs whose sites may be arranged in a variety of ways (hierarchical, centralized, decentralized, and so on). For simplicity, the distribution models discussed below are described in the context of basic point-to-point interaction between sites.

3.1 Distributed Data Source Model

In the most basic distribution model, all processing takes place at the home site (i.e., the site performing an inquiry making use of resources from other sites). Data source sharing is achieved by directly shipping data from remote sites across the network for processing at the home site. The data sources may be real-time streams, or they may originate from stored data. Implementing this distribution model creates the necessity for distributed data acquisition capabilities to identify and access remote data sources (see Section 5), and a stream processing engine that can send and receive streams remotely.

The key advantage of this basic approach is its simplicity. Data from another site may be used with home-grown processing. But this approach has the potential pitfall to use excessive amounts of processing and network bandwidth. Streams originating at a remote site may be voluminous. This is especially true in the case of primal streams which may undergo little to no processing at the remote site to reduce their size. Derived streams may be at a more manageable data rate, presenting less of an issue, but in some cases even a derived stream will be voluminous.

3.2 Distributed Processing Model

A more advanced approach moves the preliminary processing of a data source to the site where it originates. This addresses the issue of sending large amounts of data across the network. It can also reduces duplicate processing when two or more sites want to access the same data source from a third site and need to perform the same or similar processing.

This approach adds complexity, however. If a data source is not already being accessed on the remote site, then processing must be initiated there on behalf of the home site. This remote initiation of processing may raise an issue of trust between the cooperating sites, as one site is asking the other site to execute potentially arbitrary code on its behalf. The trust issue can be addressed through the CIP that exists between the sites. One aspect of a CIP would reflect the agreement each site has negotiated, by specifying which data sources it is willing to share, and what sorts of processing it is willing to perform on those data sources.

Different schemes may be used to achieve distributed processing. First, it is possible to transfer effective ownership of some resources in the remote site to the home site, letting the home site's scheduler allocate those processing nodes. This requires an extreme level of cooperation and trust between the two sites. Second, it is possible to have the processing scheduled by the remote site itself. This approach retains a greater degree of site autonomy. In addition, it may facilitate sharing if multiple sites want to access the same stream.

3.3 Distributed Planning Model

The most sophisticated distribution model considers the availability of both data sources and processing at multiple sites as part of the planning process. If the home site requires several data sources from a remote site, it may be logical to send the entire job over to the remote site. Likewise, it might make sense to break up a set of PEs among sites according to the availability of data sources and processing capability at each site.

Several considerations must be made as part of distributed planning. In order to intelligently partition a processing graph, the availability of data and processing at each site must be known. In addition, the load at each site must be understood. This implies consideration of what other jobs are running at a specific site, and how important they are in comparison to the job being planned. Finally, the execution of the distributed plan must be monitored carefully, to make sure that each site involved is operating effectively, and that the overall plan is executing as efficiently as possible across the sites. Execution issues discovered via monitoring feedback can trigger replanning of some or all of the job.

This approach is much more complex than the models described above, but it is also the most powerful. It requires support from several components of System S, including Inquiry Services and the scheduler. It also requires a much higher degree of interoperability and trust between sites, based on sites' CIPs. At the same time, distributed planning is the key to system-wide or region-wide effectiveness and efficiency. Multiple sites that wish to cooperate for the good of the whole, rather than each optimizing itself independently, can make the best use of resources by optimizing the subdivision and placement of jobs according to their inputs, execution patterns, priorities, and other factors.

Another variation on distributed planning is an even more integrated approach where the job manager and scheduler span multiple sites, allowing them to optimize multiple sites as a whole. This would require the greatest amount of interoperability and trust between sites.

4. COMMON INTEREST POLICIES

A Common Interest Policy (CIP) specifies the boundaries, both "forbidden" and "allowed", of interoperability among collaborating sites. In general, anything that is not explicitly allowed is forbidden. For rarer cases where sites have extremely high degrees of mutual trust, it can be the opposite. In addition, a CIP might explicitly specify other "forbidden" or "allowed" interoperations for subservient CIPs. (As an example, IBM might impose a CIP across all its sites that dictates a broad range of cooperation while simultaneously disallowing a wide range of cooperation with sites outside IBM.) A CIP includes such things as:

- What streams and locally stored data are remotely accessible? Since the authors of the CIP cannot predict every stream in advance, there needs to be a way to represent classes of streams. For instance, when creat-

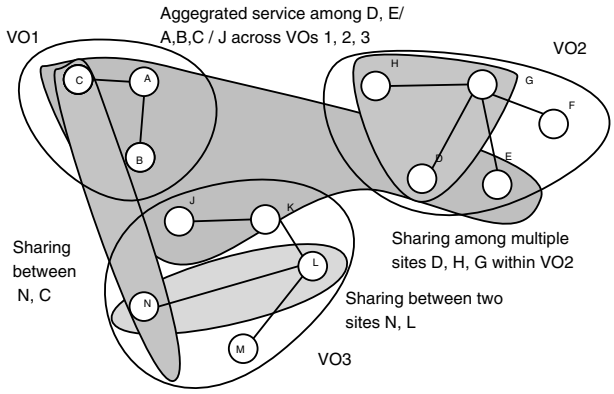

Figure 1: Representative sharing scenarios for distributed data acquisition.

ing a stream, one could tag it as globally public, locally public, or private. As another example, a CIP might specify that all streams that have been sanitized in a particular fashion are the only ones that can be shared.

- What costs, if any, are incurred by accessing remote data? This would apply to streams and stored data. The cost can be in monetary or other forms such as barter.

- What remote resources are available, and at what costs? Is the interaction model simply distributed access to data sources, distributed processing, or even distributed planning?

- What resources will one site provide another for failure recovery? When is it permitted, what types of data can be replicated, and what are the limits on replication and remote execution?

While CIPs are negotiated by humans, they are of course interpreted by computers. The format of CIPs is not yet decided, but a logical choice is to base them on XML. Since both sites and VOs are able to enter into relationships with other sites/VOs, CIPs must support hierarchical authorities with appropriate conflict resolution in the case of incompatibilities.

5. DISTRIBUTED DATA ACQUISITION

Distributed data acquisition refers to the discovery of desired data sources originating in other sites and the transportation of data from those sites. Figure 1 depicts three representative scenarios of data sharing. The first is direct point-to-point data sharing between two sites governed by a CIP (e.g., {N,L} and {N,C}). This scenario also involves the transportation of remote data to the home site. The second is data discovery among many sites belonging to the same VO (e.g., {D,E,F,G,H} in VO 2). One site needs to discover eligible data sources from potentially all the other sites. The third is an *aggregated service* between two VOs; for example, all sites within one VO might collectively provide service to sites in another VO (e.g., {A,B,C} in VO1 share up to 30% of data sources with {D,E} in VO2).

5.1 Point-to-Point Data Acquisition

This scenario is the most primitive form of data sharing; it can be used among a small number of collaborating sites. There is a CIP between each pair of sites that specifies what data sources can be shared between them, and under what conditions. When one site needs remote data sources from others, it can send requests to the other sites. The latter will check the CIP and determine exactly how they should respond to the requests, such as which data source records to return.

The discovery of remote data sources can use pull or push mode. In the pull case, the home site sends out requests to remote sites on demand. In the push case, remote sites actively send some of their data source records to the home site. To instruct remote sites which data sources they need, the home site can insert "triggers" at remote sites. These triggers specify the characteristics of desired data sources. When new or existing data sources become eligible, the triggers are executed and remote sites push such data source records to the home site. It is possible for one site to use a combination of pull and push modes.

5.2 Multiple Site Data Acquisition

When the number of collaborating sites is small (e.g., fewer than 50–100), full mesh point-to-point data acquisition between each pair of sites may be sufficient. However, we do not want to limit cooperation to this number of VOs and when the number increases, a full mesh approach is not scalable. Each site needs to manage the discovery with each other site. The addition or withdrawal of any site affects all other sites, not to mention the pair-wise discovery traffic.

To address these issues, we need a unified interface that can discover data sources from all collaborating sites. One possibility is to organize the sites into a hierarchy. Each site chooses another site as its parent. They collectively form a tree structure. The hierarchy can naturally follow existing administrative relationships within an organization that owns multiple sites. Organizational peers, which are not subordinate to each other, can negotiate among themselves and determine the hierarchy formation.

Each site summarizes its data sources in aggregated forms and sends the summary to its parent. The summary is a condensed representation of the original data source records and supports attribute-based search. It can take many different forms. For example, the lower/upper bounds or histogram can be used to summarize the DATA_RATE attributes of a site's video data sources.

A parent further aggregates the summaries from its children and itself, and sends the result up the hierarchy. In this way, summaries are aggregated and propagated bottom-up in the hierarchy. The root site will have a "global summary" of all the data sources and each site has a "branch summary" of data sources owned by its descendants.

The discovery of data sources starts from the root site. A client site sends a request to the root, which examines its own data sources and its children's summaries. It returns its eligible data sources to the client, and instructs the client to search its child branches that have matching summaries.

In this way, the client can eventually discover eligible data sources from all sites. More mechanisms such as caching and redundancy are needed for the performance and resiliency of such an approach.

5.3 Aggregated Service

When two VOs collaborate and form a larger VO, they may choose to specify services provided to each other in aggregated terms. This can happen between large organizations each of which owns multiple sites. For example, IBM might agree to share up to 30% of its data sources with Lenovo. This aggregated number is a combined contribution from all sites within IBM. IBM needs to enforce this quota so as to avoid overcommitting its resources.

In such cases, some auditing is needed to enforce the policy. One possibility is to have an Accounting Center (AC) for each VO. A site must register and update its contribution to the other VOs at the AC. Whenever a site within a VO answers a request from outside, it also checks with the AC to ensure that the aggregated terms are not violated. Otherwise it needs to transform the results, possibly returning less or no records, to stay within the quota.

5.4 Delegation

For security, privacy, trust, or management considerations, one site may choose to delegate the interfacing responsibility to another site. For instance, in a collaboration between two large VOs, sites in one VO delegate a representative for all search requests from the other VOs.

The delegation serves a number of purposes. First, it can reduce security risks. Sites other than the representative do not need to interact directly with the outside world, which can be less trustworthy than the internal VO environment. This is similar to having a firewall protecting an internal network. Second, it helps to centralize and ease the management overhead. Much of the daily operational complexity can be shifted to the delegation site, which is easier to manage than managing potentially large numbers of sites directly. For example, the AC can be located at the delegation site, which is at a suitable position to enforce the aggregated service terms. For external sites, they only need to deal with the delegation, instead of handling each of the other sites individually.

6. FAILURE RECOVERY

Failures can occur in System S in a number of ways. Individual PEs or applications can fail. Various system components, both hardware (e.g., storage and computation nodes) and software (e.g., INQ, DSM) can also fail. The failure of system components will at a minimum cause the degradation of the site's capability and at worst cause the failure of the entire site. Even partial failures of system components can dramatically impact the capacity of a site.

Failure recovery is important both *within* a site and *between* sites, but our emphasis here is on cross-site failure recovery. The interesting challenges include the mechanisms for supporting recovery and the policies governing issues such as site selection and frequency of backups.

6.1 Mechanisms

System S is intended to run effectively under overload. Many noncritical applications can be terminated under appropriate circumstances. These applications need no special support for recovery when they or the nodes on which they run fail. Applications that are more important, yet not critical, can be restarted from scratch upon a failure without significant loss to users. A relatively small but critical fraction, however, should be resumed after a failure without loss of state. For these, one can use well-understood failure recovery techniques, such as process-pairs [10] or checkpointing [11].

These techniques work well for recovering within a site. They can also be used to run critical applications on another site, either in parallel (process-pairs) or upon a failure (checkpointing); however, the overhead of maintaining the state across multiple sites will be substantially higher than within a more tightly-coupled site.

To handle failures of hardware system components, two mechanisms are available. The first is *load shedding and rebalancing* within one site. After a failure of some nodes, low-priority jobs can be killed or suspended to make room for high priority ones. High-priority jobs can also be redistributed among the remaining nodes, thus rebalancing the workload on the functioning nodes. The second is *inter-site offloading*. If the workload of important jobs in a site exceeds the capacity of the remaining nodes, the site can shift some of its high-priority jobs to other sites. The sites need to pre-arrange CIPs among them to determine which jobs to offload and how to offload them. Executing in another site faces heterogeneity in available data sources, execution environments, competing execution priorities, and other issues; thus it is a course of last resort.

In rare instances, an entire site may fail. This may be the result of a natural disaster such as floods or earthquakes, or the simultaneous failure of each instance of a critical system component. The primary difference between partial and total site failure is that in the former case, the affected site can initiate recovery actions, while in the latter case, another site must detect and respond to the failure. The choice of which site (or sites) backs up a given site must be negotiated in advance, based on their CIP(s). Critical data, such as the state necessary to run specific applications and the stored data upon which those applications rely, must be copied to the backup site(s) in advance. Any applications that are critical enough to be checkpointed periodically or run in parallel via process-pairs must be coordinated across the sites.

6.2 Policies

There is a great deal of leeway in the CIPs between sites regarding how to respond to failures. Questions include:

- Which site(s) should backup a given site? Some sites will be excluded due to an unwillingness or incompatibility. But if multiple sites are potential backups, which subset should we choose and how should we divide and assign the work? This needs to take reliability and associated costs into consideration. Furthermore,

should the assignment of backup sites be optimized by each site individually or decided for the benefit of a group of sites as a whole?

- For checkpointed applications, how often, and under what conditions, should the checkpoints take place?

- For replicated persistent data, how does value-based retention interact with the reliability of the data [6]? Each extra copy of backed-up data takes space away from a site's own data, some of which may have only one copy.

7. HETEROGENEITY

When multiple sites collaborate, each of them may have a different operating environment, in terms of the runtime environment, type system, security and privacy policies, user namespace, and other aspects. These sites have to deal with the heterogeneity to successfully interoperate.

7.1 Runtime Environment

Each System S site has its own runtime environment, including PEs, stored data, and type system. The runtime environment can vary from one site to another. Each site can have a different set of PEs, stored data, and types, with different names, formats, functions or interpretation. For example, one site uses a 5-character string for type ZipCode. Another site may not have this type, or it uses a full 9-digit ZipCode. Transformation and mapping rules and routines between sites are needed to ensure collaborative applications use PEs, stored data and types correctly across sites.

PEs, stored data, and type systems are also expected to evolve over time. Their evolutions can differ from one site to another. Since applications using different versions of the same PE, stored data, or data types can co-exist, it is necessary to maintain the evolution history of them using mechanisms such as versioning. The transformation and mapping should also handle such evolutions, both intra-site and across sites.

7.2 Security and Privacy

Collaborating sites may have the same or distinct security and privacy policies. When a single organization operates many sites, or all sites have high degrees of mutual trust and uniformity, they may adopt a single security and privacy policy and a common user namespace. System S assumes lattice-based secrecy [17] and integrity policy models. Each site will understand the format and implied relationships of security labels used by all sites; the access rights and restrictions encoded within a security label are uniformly applicable throughout all the sites.

When multiple sites belonging to different organizations collaborate, however, uniform policies may not be feasible. Each site needs to define its own security and privacy policies. All sites would have secrecy levels and confidentiality categories defined for their subjects and objects, but the numbers of secrecy levels, sets of categories, and their meaning and interpretation, can be totally different. The user name space can also vary, and even be completely separate from one site to another.

In such a scenario, policy translation and mapping are needed. For example, in a collaborative hurricane response and recovery system, IBM could use two secrecy levels (public and IBM-confidential) and no categories, whereas the Federal Emergency Management Agency (FEMA) may use four secrecy levels (unclassified, confidential, secret, top-secret) and a large set of categories, including a category IBM-NDA assigned to subjects to deal with IBM confidential information. The policy translation/mapping rules may define that IBM sites would provide IBM confidential data only to FEMA subjects cleared to at least the "confidential" level and with category IBM-NDA.

8. RELATED WORK

Although System S as a whole has many distinctive features, here we focus on the distribution and cooperation aspects of the system.

Borealis [2] is a distributed stream processing analysis system with a number of similarities to System S. It has explicit support for fault tolerance [4, 12] as well as contracts to "sell" load between sites in a federated system [5]. System S differs fundamentally from Borealis and other stream processing systems such as STREAM [19] and TelegraphCQ [7] in several aspects. First, the cooperation among System S sites encompasses a variety of interaction models, from loosely coupled to tightly integrated. They address different levels of cooperation needs of sites with varying degrees of trust relationship, and intersite heterogeneity. Second, it supports generic application-specific processing rather than database operations— a more difficult problem due to higher complexity, development costs and times to completion [18]. Finally, System S has an Inquiry Specification Language that allows users to specify application declaratively at semantic level. This is very important to allow users focus on application level tasks, rather than deal with the complexity of finding the optimum set and interconnection of data sources and PEs.

Grid computing is another area of related work. Virtual organizations arose in the context of the Grid [9] and we adopt similar constructs in our system. In addition, there has been substantial work in matchmaking between different organizations based on required capabilities (e.g., Liu, et al. [14]).

For failure recovery, the field has decades of work on techniques such as process-pairs [10] and checkpointing [11]. In this area, our emphasis is on policies such as optimizing the selection of backup sites and providing a balance between the goals of different sites, and to incorporate existing underlying failure recovery mechanisms.

9. CONCLUSIONS AND FUTURE WORK

System S is an ambitious project, and it is undergoing active design and implementation. The focus of this paper is how to realize cooperative stream processing among multiple System S sites, further increasing the scale, breadth, depth, and reliability of analysis beyond that available within a single site. We have identified the main problems and presented the general design, including the models and agreements with which sites interoperate, the methods for supporting distributed data acquisition and distributed process-

ing, support for site failover, and allowance for the heterogeneity within and across sites. As the design evolves and we gain more experience with the single-site system, we expect to add to this list and refine the design. Our ultimate objective is to realize cooperative stream processing under extreme scales, thus fully exploiting the potential of System S.

Acknowledgments

We thank Brad Fawcett, Nagui Halim, Kang-Won Lee, Zhen Liu, and the rest of the System S team for helpful feedback on the design of cooperative System S and/or earlier drafts of this document.

10. REFERENCES

[1] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Centintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag S. Maskey, Alexander Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. The design of the borealis stream processing engine. In *CIDR 2005 - Second Biennial Conference on Innovative Data Systems Research*, 2005.

[2] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag S Maskey, Alexander Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. The Design of the Borealis Stream Processing Engine. In *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, January 2005.

[3] Lisa Amini, Nevendu Jain, Anshul Sehgal, Jeremy Silber, and Olivier Verscheure. Adaptive control of extreme-scale stream processing systems. In *Proceedings of ICDCS 2006 (to appear)*, 2006.

[4] Magdalena Balazinska, Hari Balakrishnan, Samuel Madden, and Mike Stonebraker. Fault-Tolerance in the Borealis Distributed Stream Processing System. In *ACM SIGMOD Conf.*, Baltimore, MD, June 2005.

[5] Magdalena Balazinska, Hari Balakrishnan, and Mike Stonebraker. Contract-based load management in federated distributed systems. In *Symposium on Network System Design and Implementation*, March 2004.

[6] Ranjita Bhagwan, Fred Douglis, Kirsten Hildrum, Jeffrey O. Kephart, and William E. Walsh. Time-varying management of data storage. In *First Workshop on Hot Topics in System Dependability*, June 2005.

[7] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michaeeal J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Frederick Reiss, and Mehul A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Conference on Innovative Data Systems Research*, 2003.

[8] Fred Douglis, John Palmer, Elizabeth S. Richards, David Tao, William H. Tetzlaff, John M. Tracey, and Jian Yin. Position: Short object lifetimes require a delete-optimized storage system. In *Proceedings of 11th ACM SIGOPS European Workshop*, 2004.

[9] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[10] Jim Gray and Andreas Reuter. *Transaction Processing : Concepts and Techniques*. Morgan Kaufmann, 1992.

[11] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. In *Readings in Database Systems (2nd ed.)*, pages 227–242. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994.

[12] Jeong-Hyon Hwang, Magdalena Balazinska, Alexander Rasin, Ugur Cetintemel, Mike Stonebraker, and Stan Zdonik. High-Availability Algorithms for Distributed Stream Processing. In *The 21st International Conference on Data Engineering (ICDE 2005)*, Tokyo, Japan, April 2005.

[13] Navendu Jain, Lisa Amini, Henrique Andrade, Richard King, Yoonho Park, Philippe Selo, and Chitra Venkatramani. Design, implementation, and evaluation of the linear road benchmark on the stream processing core. In *25th ACM SIGMOD International Conference on Management of Data (SIGMOD 2006)*, June 2006.

[14] Chuang Liu, Lingyun Yang, Ian Foster, and Dave Angulo. Design and evaluation of a resource selection framework for grid applications. In *In Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing*, July 2002.

[15] Anton Riabov and Zhen Liu. Planning for stream processing systems. In *Proceedings of AAAI-2005*, July 2005.

[16] Anton Riabov and Zhen Liu. Scalable planning for distributed stream processing systems. In *Proceedings of ICAPS 2006*, June 2006. To appear.

[17] Ravi Sandhu. Lattice-based access control models. *IEEE Computer*, November 1993.

[18] Michael Stonebraker, Ugur Çetintemel, and Stanley B. Zdonik. The 8 requirements of real-time stream processing. *SIGMOD Record*, 34(4):42–47, 2005.

[19] The STREAM Group. STREAM: The Stanford stream data manager. *IEEE Data Engineering Bulletin*, 26(1), 2003.