Autonomic Operations in Cooperative Stream Processing Systems

Michael Branson[1] Fred Douglis[2] Brad Fawcett[1] Zhen Liu[2] Anton Riabov[2]
Fan Ye[2]

[1]IBM Systems and Technology Group, Rochester, MN USA
[2]IBM T.J. Watson Research Center, Hawthorne, NY USA

Abstract

System S is a large-scale distributed streaming data analysis environment, designed to handle extreme data rates. Multiple System S sites can cooperate to further improve the scale, breadth and depth of data analysis. We describe three autonomic features in the operation of such a cooperative stream processing environment: interoperation models, planning, and failover, They enable the distributed environment to deal with dynamic and rapid changes in imposed workload, available resources, and the priorities of administrators and users. Thus, the system can minimize the human effort needed to operate such large, complex systems.

1. Introduction

System S is a data stream processing system that enables the execution of interconnected processing tasks on continuous data feeds. It has been designed from the outset under the assumption that the system will be continually overloaded and must adapt to rapid changes in load and resource availability while addressing varying priorities of administrators and end-users [6].

A single System S *site* is a collection of processing resources under one administrative domain. It could vary in scale from a single computer (whose job is to input *primal* data streams from outside the system and pass them to other sites) to a processing complex with thousands of computing nodes, terabytes of storage, and terabits/s of network connectivity. These larger sites produce *derived* streams through data analysis, eventually returning results to users.

We have been working on the design and implementation of the architecture for these independent sites to cooperate; they can gain analysis breadth and depth not achieveable by any indidvidual site. For example, Federal Emergency Management Agency and local Department of Transportation sites can share information about emergency supply and traffic conditions for evacuation plans during a hurricane. The focus of this paper is the following three themes in autonomic operations for cooperative processing.

Interoperation models for self-configuration We generalize *Virtual Organizations* (VOs) from the Grid computing community [7] to allow sites as well as VOs to dynamically form, join and leave collaboration constructs, referred as *Virtual* Virtual Organizations (V^2Os), which allow System S sites to configure themselves as needs arise and policies adapt.

Planning for self-optimization Within one site or a V^2O, a *Planner* component takes a specification of desired end results posed by a user and creates an optimized plan that produces the desired results [13] using currently available resources within the site or V^2O. To adapt to dynamic resource availability over a long time and within large V^2Os, the planner would ultimately be able to generate event-based plan branches that define how to modify the plan automatically upon certain pre-defined events.

Failover for self-healing Sites can monitor each other, arranging to recover critical processing in the event of failures. We also envision other failover support such as periodic checkpoints and adaptive monitoring based on priorities and resource availability.

We note that the features described in this paper range widely in their maturity. Some are prototyped, some are designed, and some are only in the early stages of design. We use appropriate tenses to disambiguate.

The rest of this paper is organized as follows. Section 2 describes V^2O concepts and how they can adapt autonomically. Section 3 explains how the planner works and how the event-based plan branch model can extend it. Section 4 discusses various mechanisms for handling failures. Section 5 discusses related work and Section 6 concludes.

2. *Virtual* Virtual Organizations

The grid computing community has long defined *Virtual Organizations* (VOs) as a way of allowing resource sharing

among multiple administrative domains [7]. It enables computation that exceeds the capability of individual organizations. A VO specifies resource sharing policies that control the resources each domain (which we call a *site*) will potentially contribute to the VO.

Historically, VOs support a fairly simple and homogeneous model for sites to interact. Sites offer resources, such as computation, data, and storage capacity, with an optional cost to access the resources. Any other sites within the same VO can request access to the resources, possibly through an *agreement* that binds the two sites to a particular sharing arrangement for a specific time interval [2]. Such a VO configuration usually requires significant human efforts, which can be overwhelming when one needs VOs to further scale up for much broader and deeper analysis.

We propose an architecture that allows sites and VOs to self-configure automatically in a variety of interoperation modes. We treat sites and VOs interchangeably from the standpoint of making agreements to provide or obtain resources. Just as a site can agree to provide N processors of a certain type for a certain interval, a VO can agree to provide the resources of its constituent members.

The architecture abstracts VOs in the same way VOs abstract physical organizations, thus we call such constructs ***Virtual* Virtual Organizations**, or V^2Os. V^2Os allow already formed collaborations to cooperate with each other in their common interest. Thus the association is not limited to individual sites, but any level of existing V^2Os (which include sites and VOs, recursively). Our V^2Os will support a number of key features for the self-configuration of V^2Os, including:

- Dynamic creation of V^2Os for ad-hoc collaborations, wherein a variety of interaction models, including both federation (in which sites and V^2Os relinquish some control to a common authority) and cooperative (equal peers with no common authority) are supported. Both tightly-coupled collaborations with significant intersite data transfer and loosely related collaborations with little communication are possible.

- Scalability for V^2Os to include thousands of sites and hundreds of VOs, as well as the ability for one site to participate in hundreds of simultaneous collaborations in the context of different V^2Os.

- Flexible management of heterogeneous resources, for which the types and availability vary substantially at different sites. The system should provide sites with the ability to request exclusive control of others' resources or be granted best-effort access to resources shared with others, depending on the policies governing interaction between the sites.

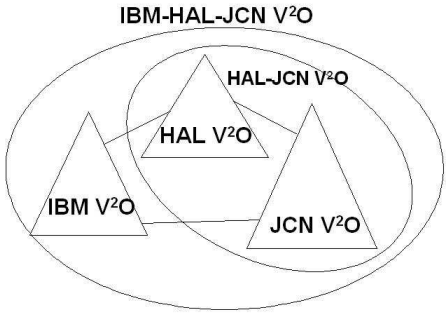

Figure 1: Cooperative V^2O for IBM, HAL, and JCN.

We have previously described the concept of *Common Interest Policies* (CIPs) [6], used to support intersite cooperation. For each V^2O, there is a CIP that specifies what resource sharing is allowed between which members. Members in a V^2O must conform to the CIP when sharing their resources. Our current prototype uses a CIP text file containing certain terms, such as:

```
SiteA;SiteB;ACCESS_RESOURCE;SHOULD;
QUALIFIER:DATA_SOURCE:
EXTENDED_QUALIFER:TYPE=CAMERA:
MONETARY_COST:100
```

This CIP defines that Site A should provide its camera data sources for Site B to access and that the monetary cost for accessing a data source is 100 units. System S has a VO management component that is responsible for managing the V^2O. The VO management components on different sites read in the CIP file and contact each other, thus forming a V^2O.

CIPs must be negotiated and agreed upon by members of a V^2O. The negotiations can be done by humans or systems. Although currently the CIPs are negotiated by administrators in our prototype, we envision that the VO management component will eventually become the entity to negotiate Common Interest Policies. Administrators will set up *site policies* that define under what kinds of conditions a site should form what V^2Os with which other sites or V^2Os. This will enable them to choose the configuration most suitable for their administrative requirements and resource availability.

Consider an example (see Figure 1): Companies IBM, HAL, and JCN form a V^2O encompassing their respective V^2Os. When a fourth company (KDO) wants to establish a cooperative relationship with IBM-HAL-JCN, it may select among multiple configurations:

- It can negotiate to create a new V^2O, with two V^2Os within it: KDO and IBM-HAL-JCN. This is suitable when KDO does not have as high trust levels as the three sites already have within the IBM-HAL-JCN V^2O.

- It can negotiate to join IBM-HAL-JCN, forming a new IBM-HAL-JCN-KDO V^2O. Each company within this V^2O will have the same rights and trust levels to the other companies.

- It can negotiate to have sites within the KDO V^2O join a new V^2O with the other companies, if it allows those individual sites to have more autonomy to determine their resource sharing in the new V^2O.

3. Planning

In System S, a job is an execution unit that accomplishes certain tasks through stream analysis. A job takes the form of a processing graph, consisting of data sources and processing elements (PEs) interconnected in a certain manner. PEs are software components that take certain input data streams and process the data in some manner, many times using a specific algorithm, and produces output data streams that can be consumed by other PEs.

System S has two self-optimizing features regarding jobs, a planner [13] and event-based plan branches. Due to the potentially large numbers of data sources and PEs shared among many users, a processing graph may be quite complex. There may exist many different processing graphs, all of which can accomplish a given goal, but with various performance/cost tradeoffs. It is infeasible for human users to manually construct and identify the best alternative. To this end, we let the system optimize itself: a *planner* component can construct an optimized processing graph autonomically, based on high-level descriptions of the desired results from the user. We call such descriptions *inquiries*; they describe the semantic meaning of the desired results.

To formally describe the semantics, we utilize *ontologies* [12], a standard from Semantic Web, to represent the concepts and relations relevant to a certain domain of interest. A concept is the abstraction for a kind of entities sharing common characteristics. For example, for emergency response applications, Sensor, Location and MultimediaData are concepts. There are specific entities for each concept. Traffic Camera 10036-1 is an entity of concept Sensor, and BwayAt42nd (the intersection of Broadway and 42nd Street) is an entity of concept Location. Concepts are associated with each other through properties that describe the relationship between them. Sensor and Location are related to each other through property atLocation, meaning that a sensor is located as a certain location.

Using the concepts and relations defined in the ontologies, we can describe data sources, PEs and users' inquiries. Data sources are described by the semantics of the data objects they generate; PEs are described by the semantics of data objects they consume and produce; inquiries are expressed by the semantics of end results users desire. An AI planning algorithm [13], enhanced to utilize such semantic descriptions, automatically composes appropriate data sources and PEs together into jobs that answer users' inquiries.

In our prototype, the planner can work in both the single-site and V^2O environments. In each case it utilizes information about available data sources and PEs and then composes plans out of them. The CIP of a V^2O specifies what kinds of data sources and PEs each site should contribute to the V^2O. A VO planner collects the semantic description about such data sources and PEs, and produces plans using resources from the whole V^2O. Based on factors including resource availability (e.g., which site has what data source, PE, or special processing hardware; the network connectivity and bandwidth between any two sites; or which sites are alive), the VO planner generates an optimized plan that uses available resources and partitions the plan among available sites. The result is a distributed job consisting of multiple subjobs, each of which is dispatched to run on one member site.

In a large V^2O, there are significant variations in resource availability over long time periods and on different member sites. They affect not only the partitioning of plans, but also what plans are feasible at all. A plan that is produced and optimized initially may become inefficient (e.g., when the quality of some data source deteriorates), or completely infeasible (e.g., when communication between two sites is lost) over time.

To allow the system adapt to dynamic environment, we propose *event-based plan branches*. (These are like contigency plans in the AI planning literature [8], but specialized to the streaming context.) Instead of requesting a panacea plan, the user defines a set of contingency scenarios with which he or she is most concerned and wants to be prepared. These are events that are very likely to happen (such as a network connection known to be unreliable), or their occurrence will have a severe impact on the application (such as a camera no longer providing relevant data because a hurricane has moved out of its range). The system can employ specialized PEs to detect and publish these events.

Under each scenario, there is certain resource availability. The planner will produce an optimized plan as a "branch" under that scenario. It would also specify the conditions under which a given plan branch should be activated. The events and their plan branches would be submitted to a Remote Execution Coordinator component. It would subscribe to events published by those specialized detection PEs and execute different branches accordingly.

Depending on the constraints such as how quickly the system must adapt, plan branches have several options for advance preparation. They range from pre-deploying and activating branches to run when the event is triggered, to

deploying them on demand, or replanning when an unexpected scenario actually happens. In this way, the whole V^2O can self-optimize a long-running job to adapt not only to events that alter resource availability during its lifecycle, but also to user constraints on how to adapt.

4. Failover

Failover in System S has been described by Rong, et al. [14]. Because it appears elsewhere, and due to space limitations, we limit this discussion to a brief summary.

The key requirements for failover support are the ability to self-heal, such as monitoring the liveliness of another site and migrating executing PEs from a failed site to a backup site (these two features are implemented in our current prototype). The PEs may interact with only streams on the failed site, or with remote streams that come from or go to other sites. In the latter case, the system must not only reinstantiate the failed PEs on another site, but also reconnect the streams that cross site boundaries.

It is possible that data objects have already been transmitted on a stream that goes to a failed site and get lost. For completely reliable stream transport, the site transmitting those objects must buffer the objects until the application consuming them has been checkpointed and migrated to a backup site. In general, there is a large body of work on checkpointing in message-based systems, which our system can leverage.

One self-healing scenario the system must address is when an executing application cannot be recreated on a new site. For instance, it might draw on data only available on the failed site, use site-specific hardware, or need other specialized resources. In this case, upon the detection of a failure, the inquiry responsible for the failed tasks can be replanned and deployed using available resources, or the alternative plan can be prepared in advance as an event-based plan branch, triggered by the failure of one of the sites involved.

5. Related Work

Cooperative System S has some commonality with other streaming data analysis [1, 4, 15] systems, Grid computing [7], and parallel execution engines [9, 5]. A representative stream processing system is Borealis [1], which has explicit support for fault tolerance [3].

Nevertheless, System S differs from Borealis and other stream processing systems [15, 4] in several aspects. First, the V^2Os allow System S sites to self-configure and cooperate under 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 relationships. Second, System S adopts the planning approach to self-optimize jobs without explicit human intervention. Users need to specify only the desired results without manually constructing jobs. The event-based plan branches will further allow the cooperative processing environment to self-optimize the runtime execution of jobs to adapt to dynamic changes in load and resource availability. This is very important for long-running stream applications to adapt to dynamic changes that cannot be predicted beforehand. Finally, System S supports arbitrary application-specific processing rather than database operations— a more difficult problem due to higher complexity, development costs and times to completion.

The closest work to our V^2O is hierarchical VOs presented by Kim and Buyya [11]. They define a VO as a set of users, resource providers, and sub-VOs, which can in turn consist of other users, resource providers and sub-VOs. In each VO, there are policies associated with the resource providers to specify the amount and cost of resources that may be consumed by the users.

Compared to us, this work is limited to resource provider lookup and only in hierarchical VOs; it does not provide the more advanced features that we propose. First, only hierarchical VOs are possible; a cooperative interaction model when there exist multiple peer VOs without a common root is not supported. Second, one VO can participate in only one higher level VO, it cannot be members of multiple simultaneous collaborations. Finally, all sites within a VO are treated identically and no differentiation is possible. These all constrain the breadth and variety of possible collaborations, and types of applications.

Various aspects of System S have been described in the literature. The most relevant to this paper include a high-level overview of cooperation in System S [6] and a recent evaluation of autonomic support for job management in a single System S site [10].

6. Conclusions

Large-scale stream processing is a grand challenge. Making multiple such stream processing sites cooperate to further scale up the analysis breadth and depth is even more ambitious. The mere management demand needed to allow such sites to operate and adapt to dynamically changing environments may already overwhelm users and administrators.

In this paper we have described three autonomic features that will relieve humans from much of the huge burden. The flexible V^2Os allow sites to self-configure and form cooperation relations suitable for their common interests and trust levels. The system uses a planner to self-optimize streaming jobs from high level descriptions from users; different plan branches can be executed upon certain events to self-optimize the runtime operation. Failover

support enables jobs on failed sites to be migrated to other sites, and even trigger autonomic replanning for improved performance. These features minimize explicit human intervention needed to deal with dynamic changes in a large, complex environment.

Acknowledgments

We thank the anonymous reviewers for feedback on an earlier version of this paper, and the rest of the System S team for their efforts on the system.

References

[1] D. J. Abadi et al. The Design of the Borealis Stream Processing Engine. In *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, January 2005.

[2] A. Andrieux et al. Web Services Agreement Specification (WS-Agreement), Version 2006/07. GWD-R (Proposed Recommendation), Grid Resource Allocation Agreement Protocol (GRAAP) WGGRAAP-WG, July 2006.

[3] M. Balazinska et al. Fault-Tolerance in the Borealis Distributed Stream Processing System. In *ACM SIGMOD Conf.*, Baltimore, MD, June 2005.

[4] S. Chandrasekaran et al. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Conference on Innovative Data Systems Research*, 2003.

[5] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI'04)*, pages 137–150.

[6] F. Douglis et al. Multi-site cooperative data stream analysis. *Operating System Review*, 40(3):31–37, 2006.

[7] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[8] J. Hoffmann and R. Brafman. Contingent planning via heuristic forward search with implicit belief states. *Proceedings of Fifteenth International Conference on Automated Planning & Scheduling (ICAPS-05)*, pages 71–80, 2005.

[9] M. Isard et al. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2007.

[10] G. Jacques-Silva et al. Towards autonomic fault recovery in system-s. In *Proceedings of the 4th IEEE International Conference on Autonomic Computing*, June 2007. To appear.

[11] K. Kim and R. Buyya. Policy-based Resource Allocation in Hierarchical Virtual Organizations for Global Grids. *Proceedings of the 18th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'06)-Volume 00*, pages 36–46, 2006.

[12] D. McGuinness and F. van Harmelen. Owl web ontology language overview. In *W3C Recommendation*, 2004.

[13] A. Riabov and Z. Liu. Scalable planning for distributed stream processing systems. In *Proceedings of ICAPS 2006*, June 2006.

[14] B. Rong et al. Failure recovery in cooperative data stream analysis. In *Proceedings of the Second International Conference on Availability, Reliability and Security (ARES 2007)*, Vienna, Apr. 2007.

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