Clusterken: A Reliable Object-Based Messaging Framework to Support Data Center Processing

Marc Stiegler, Jun Li, Karthik Kambatla, Alan Karp

HP Laboratories
HPL-2011-44

Keyword(s):

data center, publication/subscription, reliable object-based messaging

Abstract:

Hadoop enables high productivity in the development of MapReduce applications. However, Hadoop becomes less effective the further an application's natural pattern of computation is from MapReduce. This paper presents Clusterken, a reliable, object-based messaging framework to support data-center-based applications. Clusterken provides object-level virtual cluster management, exactly-once message processing, composable reliability, output validity, and authorization based access control. Together, these features simplify programming and improve productivity on distributed applications by enabling arbitrary interaction patterns. We compare two implementations of a specification for a publication/subscription system, one in Hadoop, the other in Clusterken. The comparison suggests that, for at least this one application, Clusterken can yield a fourfold increase in programmer productivity. Keywords: reliable object-based messaging, composable reliability, virtual cluster management, output validity, authorization-based access control, publication/subscription.

External Posting Date: April 6, 2011 [Fulltext]   Approved for External Publication
Internal Posting Date: April 6, 2011 [Fulltext]

Clusterken: A Reliable Object-Based Messaging Framework to Support Data Center Processing

Marc Stiegler
HP Labs
Email: marc.d.stiegler@hp.com

Jun Li
HP Labs
Email: jun.li@hp.com

Karthik Kambatla
Purdue University
Email: kkambatl@cs.purdue.edu

Alan Karp
HP Labs
Email: alan.karp@hp.com

Abstract—**Hadoop enables high productivity in the development of MapReduce applications. However, Hadoop becomes less effective the further an application's natural pattern of computation is from MapReduce. This paper presents Clusterken, a reliable, object-based messaging framework to support data-center-based applications. Clusterken provides object-level virtual cluster management, exactly-once message processing, composable reliability, output validity, and authorization based access control. Together, these features simplify programming and improve productivity on distributed applications by enabling arbitrary interaction patterns. We compare two implementations of a specification for a publication/subscription system, one in Hadoop, the other in Clusterken. The comparison suggests that, for at least this one application, Clusterken can yield a fourfold increase in programmer productivity.** **Keywords: reliable object-based messaging, composable reliability, virtual cluster management, output validity, authorization-based access control, publication/subscription**

I. Introduction

Hadoop [1] is very effective for developing applications that naturally transform into MapReduce problems that can be expressed with a spawn/gather distributed interaction pattern. For such programs Hadoop is used primarily as a MapReduce [10] job scheduler with automated restart, along with its reliable file system HDFS to hold both the input and output of the MapReduce job. However, the limitations of supporting only one, fixed distributed interaction pattern quickly become apparent as one explores applications for which spawn/gather is less natural. Some alternative patterns, such as stream-based dataflow processing that requires nodes to synchronously interact with each other, are so self evidently important that they have inspired development of point solutions based on Hadoop [9]. However, it is unlikely that this approach will ever allow the expression of the wide variety of patterns of computation needed for general purpose, data center computing.

One kind of application that does not fit neatly into the spawn/gather framework is publication/subscription. While details vary, the common foundation is a set of topics for which there may be publishers who add events to a topic, and subscribers who see the events for the topic; the topic will often also maintain an archive of events that can be viewed by late subscribers. A way to use Hadoop for this problem is to run the system as a series of batches: collect all the published events for a (heuristically chosen) period of time, and process that batch to deliver the events to their respective topic archives and subscribers. There are several problems, starting with the fact that Hadoop is basically a batch-oriented application execution engine, whereas the problem has a streaming flavor.

This paper presents Clusterken, which is a reliable object-based messaging framework to support data-center-based distributed applications. Clusterken provides object-level virtual cluster management, exactly-once message processing, composable reliability, output validity, and authorization based access control. Together, these features simplify user programming and improve productivity on distributed processing by enabling arbitrary interaction patterns. To demonstrate, we compare productivity when implementing a pub/sub system using Hadoop (with HBase [2] as the data store) versus in Clusterken. The comparison suggests that, for at least this one application, Clusterken can yield a fourfold increase in productivity.

II. Overview of Clusterken

A. Waterken Underpinnings

Clusterken is based on the Waterken distributed programming platform [8], which delivers several features that benefit the cluster programmer:

1) The *vat* is the basic unit of concurrency in Waterken. Vats are lightweight containers that interact with other vats only by exchanging non-blocking messages. A single vat may contain many objects; a single server may contain many vats.
2) *Promises* [15], [16] are used to deliver answers during cross-vat communication. The application programmer performs ordinary-looking object invocations to send messages between objects in different vats. The sender immediately acquires a local promise for the result and goes on to the next statement for execution without being blocked waiting for the result. Since the type of the object that will be returned is known, it is possible to invoke methods on a promised object even before the promise resolves. Observers can be set on a promise that will fire upon resolution. The vat never blocks or waits for a promise fulfillment.
3) *Turn based checkpointing* is performed transparently on a per-vat basis. Each vat has a single thread for executing application code, a queue of incoming messages for each source, and a queue of outgoing messages for each

destination. Computation in a vat begins by removing a message from one of the incoming queues, and it continues until the method invoked by the message has computed the answer to be returned. At that time, the Waterken infrastructure automatically checkpoints the state of any changed objects, the value to be returned, and the outgoing message queues as a single atomic transaction [21]. The outgoing messages and return value are released to the network only when the checkpoint is complete, giving the system the advantages of checkpoint-on-send [6] without requiring a checkpoint for each send. A turn is the interval from the delivery of the message to the target object to the completion of the checkpoint.

4) *Exactly once message processing* is enforced in conjunction with the turn based checkpointing. Each outgoing message is resent by the infrastructure until receipt of an acknowledgment from the recipient denoting completion of the checkpoint from processing the message. The message contains an identifier so that the recipient's infrastructure can distinguish a duplicate message, in which case the already-computed answer is immediately returned without processing the message a second time.
5) *Webkeys* [20] embody object references for messages crossing the network. Webkeys are specially formulated urls that authenticate the receiver, authorize the sender, and encrypt the channel. They enable easy enforcement of authorization based access control [14].
6) The *redirectory* system allows a Waterken server find another server to which it needs to send messages even when the recipient moves between nodes. Hence, a vat on a failed node can be relaunched on any node and the system will transparently reconnect objects that have references to one another.

Each of these features has important implications for productive programming of data center applications.

Direct invocation of remote objects and the ability to invoke methods on unresolved promises means that much of the code looks sequential [22]. Since an object can only invoke objects to which it has references, and the references are represented over the wire as unguessable webkeys, access is effectively controlled by limiting which objects get references to which other objects.

Non-blocking messages preclude deadlock. No other application activity occurs in a vat while it is processing a single message. Hence, there is no plan interference [17], and there are no fine grain data races. Parallelism comes from spawning more vats on each cluster machine.

Since reliable checkpointing and message delivery are transparent, the programmer need write no code to handle exception conditions caused by server crashes or network partitions. Since the checkpoints are local and independent, the program is not constrained to be fully deterministic.

Integrating the checkpointing with the messaging ensures *output validity* [13]: regardless of system failures, any state the system arrives at is a state that the system could have arrived at if different messages had simply been processed more slowly and possibly in different order.

Output validity as implemented in Waterken extends across subsystems to yield *composable reliability*: two independently written subsystems of vats have the same reliability properties when they interact as they do when executing separately [13].

The most recent checkpoints of all the vats represents a recovery line, which improves scalability because no coordination is needed for either checkpoint or recovery.

B. Clusterken Enhancements

Waterken provides a framework to support object-based reliable messaging. However, in a cluster programming environment, further infrastructure support is required to connect machine nodes to form a processing network, to specify how application code will be hosted in each node within this network, to monitor and respond to runtime node failure, and to balance the processing loads on the nodes. To meet such needs, we have built an additional layer of abstraction on top of Waterken, which we call *Clusterken*.

To enable the application to create and connect all its objects and vats throughout the cluster, Clusterken offers an object-level interface for the authorized nodes. Clusterken provides a means to define separately controllable virtual clusters (Figure 1). A virtual cluster is the subset of nodes on which an application (or application subsystem) has been granted execution privileges. These virtual clusters are lightweight, easy to create, easy to use, and easy to revoke. Both subclusters (with fewer nodes) and superclusters (composed by aggregating other virtual subclusters, possibly taken from multiple hardware clusters) can be manufactured.

In the basic layout, each OS instance (a hardware node or a virtual machine) has a Primary vatmaker; these primaries are linked to each other to form the root virtual cluster. Given a (webkey) reference to any vatmaker in a virtual cluster, one may create new vats on any OS instance in that virtual cluster. One may also manufacture a new, separately-revocable virtual subcluster with vatmakers on any subset of the parent virtual cluster. The system supports full rich sharing [19], allowing the delegation of the authority to make attenuated delegations, shifting the bulk of the administrative burden from the IT administration staff to the project managers who have local knowledge of which people and which applications need what resources. In a simple scenario, the administrator with access to the primary vatmakers could create a new virtual subcluster for the pub/sub product owner, who would then create one new virtual subcluster for the prototype pub/sub system. He would share the webkey to the subcluster with all the people on the product team, allowing those people to spawn vats on any of those hosts by specifying the root class of the code to be run. One revokes the subcluster when the prototype is no longer needed.

The Clusterken API allows objects to create new vats containing new objects on any node included in the application's virtual cluster. Arbitrary distributed interaction patterns are easily built with this facility. It takes 14 lines of code to set

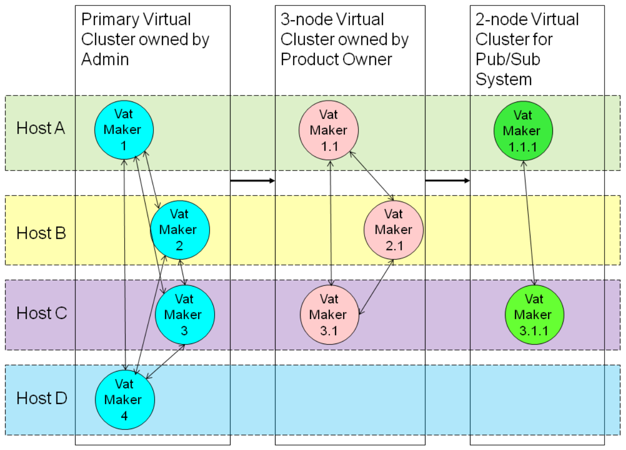

Fig. 1. The administrator starts with four Vat Makers running on four hosts, either physical or virtual. Three of these Vat Makers are cloned to create a Virtual Cluster for the Product Owner, who in turn creates a Virtual Cluster consisting of two Vat Makers, and gives a webkey to one of these Vat Makers to the person responsible for managing the pub/sub prototype (the notation 3.1.1 denotes the subsubcluster's Vat Maker on Node 3). Note that each Vat Maker in a Virtual Cluster holds references to the other Vat Makers in the same Virtual Cluster. Hence, anyone with a webkey to Vat Maker can create vats on any of the hosts in the virtual cluster.

up a traditional spawn/gather pattern of interaction, and 16 lines of code to set up a streaming ring pattern. All objects with interfaces exported from a vat can act as clients, servers, and peers. Whereas Hadoop treats all nodes as stateless compute resources, Clusterken vats are stateful. Consequently, the strategy for reliability in the presence of failure is quite different. With Hadoop, if a node ceases to respond promptly, the Hadoop scheduler restarts the last assigned computation. With Clusterken, a controller relaunches the nonresponsive vat from its last checkpoint.

Clusterken also supplies other facilities for the cluster programmer. The FarFile API provides a reliable file system abstraction integrated with the checkpointing protocol. It allows remote writes to a file, guarantees exactly-once appending to the file, and implements authorization based access control.

III. Publication/Subscription System

We have written a detailed specification for a scalable topic-based publication/subscription system [11]. An administrative facility is responsible for creating and deleting topics managed in the system. The admin also grants and revokes publication and subscription rights on a per-topic basis. At runtime, an event is published to the associated topic. The system enforces the event publication rights, matches the published event against the user subscription table and generates a notification list including the users that show interest in the events. The system then delivers the notification reliably to the identified users. Historical events and notification results are archived in a persistent store for auditing purposes.

This specification (minus the access control aspects) was implemented using Hadoop MapReduce, with HBase as the persistent data store for events, topics, user subscriptions and intermediate user notification results. In this implementation, events published within a time window (say, 20 seconds) are buffered, and then become the input to the MapReduce job, which is basically the table join between the event table represented as $\langle event, topic\rangle$ and the user subscription table $\langle user, subscription\rangle$, and produces the user notification table $\langle user, notification\rangle$. There is a second Map/Reduce job to distribute the notification messages to the designated users reliably (via multiple retries in case failures occur). These two MapReduce jobs run concurrently and independently, with the second job taking the output of the first job as its input. We chose the Hadoop Fair Scheduler to schedule these two jobs.

We have built a similar scalable system in Clusterken that meets the same specification.

IV. Implementing Pub/Sub in Clusterken

The Clusterken implementation of the pub/sub system is an object-oriented system in which the objects generally correspond to the objects in the conceptual system (see Figure 2). Except as otherwise noted, an object of any of these types resides in its own vat, and these vats are distributed across all the nodes in the cluster.

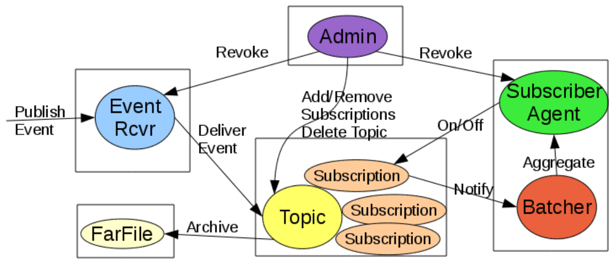

Fig. 2. A minimal Clusterken pub/sub system with one topic, one publisher, and one subscriber (multiple subscriptions are shown, going to other subscribers). Each square box represents a separate vat. These vats are distributed randomly across the nodes of the cluster. The Admin object holds references to everything that is separately revocable. The Admin object also holds a reference to a VatMaker on the virtual cluster (not shown), with which it can manufacture new vats on any node to add new pub/sub objects.

An EventReceiver object gives one publisher authority to publish events to one topic. The EventReceiver, upon receiving an event from its publisher, checkpoints the event along with the message that sends the event to the Topic. Since the publisher is outside the Clusterken system of composable reliability, special handling is needed for event submission failures. Consequently, if the publisher loses the connection to the EventReceiver during submission of an event, the publisher can retrieve the most recent successfully submitted event to see where publication left off; the publisher handles failed submissions according to the publisher's recovery scheme. (If the publisher is a person, he would look to see if the last successful submission was the last one he tried to submit.)

The Topic object is the central receiver and distributor of events for a single topic. Each topic resides in its own vat, and the Topic objects are distributed across all the nodes in the cluster. Every EventReceiver for the topic holds a remote reference to the Topic. Each Topic holds references to a FarFile (acting as an archive) and to each Subscription to the topic from a SubscriberAgent.

Each Subscription object gives one subscriber access to the events flowing through one Topic. The Subscriptions are collocated in the vat with their Topic. When an event arrives at a Topic, it notifies all the Subscriptions, each of which forwards the event to its respective Batcher/SubscriberAgent pair. Being collocated with the Topic, no communication or checkpoints are needed for the subscription notification. The messages to all the Batcher/SubscriberAgents are collected during a single turn, and there is a single checkpoint performed after notifying all the Subscriptions. That checkpoint holds the messages to all the Batchers.

A FarFile is used directly as the archive for the topic. To give an entity read access to the archive, one hands over a separately revocable read-only FarFile webkey.

The Batcher and SubscriberAgent represent the subscriber in the system. Events from all the Topics to which the subscriber has subscriptions are sent to the Batcher, which delivers them to the SubscriberAgent when there are enough events to form a batch.

There is one Admin object whose webkey is given only to the system administrator. The Admin object allows the administrator to add and remove Topics, EventReceivers, individual Subscriptions, and Batcher/SubscriberAgents.

In the Clusterken pub/sub system all the data (except the FarFile archive) is stored implicitly and automatically as part of the vat checkpoints.

The object-level virtual cluster management system enables easy distribution of vats across the cluster as new topics, publishers, and subscribers were added. A newly created object (such as a new EventReceiver or a new Topic in the pub/sub application) is assigned to a new vat on a randomly selected node in the virtual cluster, distributing the new vats smoothly across the cluster.

V. Productivity Comparison to the Hadoop-based Implementation

Table I describes interesting differences between the features and effort required for the Hadoop and Clusterken implementations.

TABLE I
Hadoop vs. Clusterken

Key Features	Hadoop	Clusterken
Effort	9 weeks	1 week
Lines of Code	1527	394
Access Control	No	Yes
Node failure handling	Automatic	Manual
Reliable messaging	Limited	Automatic
Concurrent Processing	Fair Scheduler	Randomized Vats
Tuple Matching	By table join	Output Validity
Event Processing	Batched	Streamed
Real-time	No	Soft
Pipeline Processing	No	Yes

The Effort metric, while interesting, is distorted because the Hadoop effort includes the time to design and formulate the specification (for example, on how to conform to the Amazon Simple Notification Service APIs). The Clusterken effort started with the specification already defined. The Lines of Code metric, which does not suffer the distortions of the Effort metric, still suggests roughly a factor of four increase in productivity during implementation of the application.

The Hadoop team did not have time to implement the specified access control. The Clusterken team did. Access control was achieved with almost no additional code by simply managing the visibility of references. For example, a publisher receives authority to publish events in a topic by being given a webkey to an EventReceiver. The EventReceiver interface exposed to the publisher has no method that returns its private Topic reference to the publisher. Since webkeys authorizing access to Topics are unguessable, the publisher has no way to access to the Topic directly.

For fault tolerance on node failure, Hadoop used MapReduce based node relocation and retry. The current prototype of Clusterken requires manual detection of a nonperforming node and manual relaunch of the vats on that node from their most recent checkpoints, either by rebooting the stalled node or by relaunching the associated Clusterken server on another node. Future work includes automating Clusterken's identification and management of such failed nodes and vats.

Fault tolerance for inter-node messaging is limited in Hadoop to checkpointing the output of the reducer, which can be fed as the input to the mappers of the next staged job. This was inadequate to reliably ensure correctness. If internal state is modified within a Map or Reduce task execution, it is the responsibility of the task to guarantee such state consistency when the task is re-executed, which is not trivial to implement in user-defined Map or Reduce tasks. For example, a node failure during the movement of an event from the input event queue to the archive event queue could result in event duplication because HBase has no transactional support that spans multiple rows. With Clusterken's output validity guarantees, the moment an event was checkpointed in the Event Receiver, it was guaranteed that the event would (eventually) appear exactly once in the archive and in each subscriber's list.

Hadoop used its Fair Scheduler for concurrency. In Clusterken, the concurrency is achieved by distributing the concurrent vats for individual EventReceivers, Topics, FarFile archives, and SubscriberAgents at random across the cluster.

Event Processing was handled in batches by the Hadoop system. In Clusterken, when an event came in, it was immediately checkpointed and began to stream through the system.

The batching of events meant that Hadoop could not operate as a real time system. Clusterken's operation could be thought of as a soft real-time system. On a moderately loaded Clusterken pub/sub system with a subscriber batch size of 1, the time from input of an event by a publisher to delivery to a subscriber was measured to take about 0.4s.

Pipelining with Hadoop could only be simulated via cascaded Hadoop jobs. In Clusterken pipelining was continuous, built into the Clusterken fabric. With true pipelining, stream processing is enabled. A new event can be received by the

EventReceiver stage while simultaneously an earlier event is being processed at the topic matching stage.

VI. Related Work

SpringSource [4], a large *ad hoc* collection of tools for programming cloud services, includes the RabbitMQ [3] reliable messaging system. RabbitMQ follows the AMQP protocol [5]. RabbitMQ guarantees delivery of messages sent through its queues. Messages and acknowledgments can be grouped into transactions. However, RabbitMQ is strictly a message handling system, and is not transactionally integrated with the sender's checkpoint. If RabbitMQ sends a message, and the sender crashes before its next checkpoint succeeds, a relaunch of the sender from its prior checkpoint will send an identical message that RabbitMQ is unable to recognize as a duplicate. Hence, while RabbitMQ guarantees at least once delivery, it cannot guarantee exactly once delivery. RabbitMQ does not implement output validity, which requires an atomic checkpoint of server state and outgoing messages.

MapReduce Online [9] is an attempt to modify the batch-oriented Hadoop MapReduce to support continuous query and stream processing. It changes the communication pattern between the Map tasks and the Reducer tasks such that the Map task pushes the processing results to the corresponding Reducer task. In the traditional MapReduce, a Reducer task pulls the corresponding Map task's processing results. However, due to MapReduce's intrinsic nature, wherein a Reducer cannot produce its final results until all the involved Map tasks have completed and delivered their results to the Reducer tasks, MapReduce Online is only effective in certain categories of data processing. One such category is online aggregation, in which the Reducer tasks can continuously make progress on the incomplete data from the Map tasks. In contrast, Clusterken's support of arbitrary distributed interaction patterns means there is no such data dependency constraint.

With respect to parallel data processing, Microsoft Dryad [12] allows a general data flow processing network to be described and executed on a machine cluster, making it more flexible than Hadoop. Like Hadoop, Dryad only supports batch processing without reliable messaging support for inter-node communication. Nephele [7] is more focused on a parallel data flow processing network associated with parallel database queries. Similar to Dryad and Hadoop, it is batch-oriented. Little in the way of fault tolerance has been developed for its runtime support. Flux proposes a fault tolerance architecture based on primary/secondary nodes to allow reliable execution of parallel query operations on each node [18].

VII. Conclusion

We have built a well-specified publication/subscription system twice, once using Hadoop and once using Clusterken. Comparing the two, there are a number of interesting differences. In particular, the Clusterken version

1) is streaming, while the Hadoop version is batch oriented,
2) needed one fourth the number of lines of code,
3) implemented the specified access controls, and
4) provided higher reliability, for example, avoiding event duplication.

Hadoop and Clusterken have very different performance characteristics, which become more important when we take failures into account. We plan to do performance studies to quantify these differences. Also, the MapReduce paradigm is of limited expressiveness, so it is harder for programmers to write alternate versions of the program with different performance characteristics. That's not the case for Clusterken. It's not clear that the most direct way to write a program results in adequate performance. We plan to explore such issues.

Clusterken is missing a component comparable to the Hadoop scheduler, so we are making that a priority for our next version. While the Hadoop scheduler assumes that a node is dead if it doesnt respond in a timely manner, we have more flexibility with Clusterken. For example, it is simple to restart a vat following a process crash or even an OS panic. More problematic are hardware failures, because we need access to the vats persistent checkpoint. One solution, using a shared file system, may have unacceptably high latency, so we plan to look at other approaches, such as dual ported disks.

References

[1] Hadoop. http://hadoop.apache.org/.

[2] Hbase. http://hadoop.apache.org/hbase/.

[3] Rabbitmq. http://www.rabbitmq.com/.

[4] Springsource. http://www.springsource.org/.

[5] *Advanced Message Queuing Protocol (AMQP) Specification*. 2008. http://www.amqp.org/confluence/download/attachments/720900/amqp0-9-1.pdf.

[6] J.F. Bartlett. A non stop kernel. In *SOSP*, 1981.

[7] D. Battre, S. Ewen, F. Hueske, V. Markl O. Kao, and D. Warneke. Nephele/pacts: a programming model and execution framework for web-scale analytical processing. In *SoCC*, 2010.

[8] Tyler Close. Waterken server. http://waterken.sourceforge.net/.

[9] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *NSDI*, 2008.

[10] J. Dean and S.Ghemawat. Mapreduce: simplified data processing on large clusters. *Comm. ACM*, 51, 2008.

[11] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys(CSUR)*, 35, 2003.

[12] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *EuroSys*, 2007.

[13] T. Kelly, A. H. Karp, M. Stiegler, T. Close, and H. K. Cho. Output-valid rollback-recovery. Technical Report 155, HP Labs, 2010.

[14] J. Li and A. Karp. Access control for the services oriented architecture. In *ACM Workshop on Secure Web Services*, 2007.

[15] Barbara Liskov and Lubia Shrira. Promises: Linguistic support for efficient asynchronous procedure calls in distributed systems. In *PLDI*, 1988.

[16] M. S. Miller, D. E. Tribble, and J. Shapiro. Concurrency among strangers. *Trustworthy Global Computing*, 2005.

[17] Mark S. Miller. *Robust Composition: Towards A Unified Approach to Access Control and Concurrency Control*. PhD thesis, Johns Hopkins, 2006.

[18] M. A. Shah, J. M. Hellerstein, and E. A. Brewer. Highly-available, fault-tolerant, parallel dataflows. In *SIGMOD*, 2004.

[19] Marc Stiegler. Rich sharing for the Web. Technical Report 169, HP Labs, 2009.

[20] Marc Stiegler. Towards fearless distributed computing. Technical Report 258, HP Labs, 2009.

[21] Marc Stiegler. A reliable and secure application spanning multiple administrative domains. Technical Report 21, HP Labs, 2010.

[22] Marc Stiegler and Jing Tie. Introduction to waterken programming. Technical Report 89, HP Labs, 2010.