Borealis-R: A Replication-Transparent Stream Processing System for Wide-Area Monitoring Applications[*]

Jeong-Hyon Hwang, Sanghoon Cha, Uğur Çetintemel, and Stan Zdonik
Department of Computer Science, Brown University
115 Waterman Street, Providence, RI 02912, USA
{jhhwang, scha, ugur, sbz}@cs.brown.edu

ABSTRACT

Borealis-R is a replication-based system for both fast and highly-available processing of data streams over wide-area networks. In Borealis-R, multiple operator replicas send outputs to downstream replicas, allowing each replica to use whichever data arrives first. To further reduce latency, replicas run without coordination, possibly processing data in different orders. Despite this flexibility, Borealis-R guarantees that applications always receive the same results as in the non-replicated, failure-free case. In addition, Borealis-R deploys replicas at select network locations to effectively improve performance as well as availability.

We demonstrate the strengths of Borealis-R using a live wide-area monitoring application. We show that Borealis-R outperforms previous solutions in terms of latency and that it uses system resources efficiently by carefully deploying and discarding replicas.

Categories and Subject Descriptors

H.2 [**Database Management**]: Systems

General Terms

Design, Experimentation, Management, Performance, Reliability

Keywords

availability, fault tolerance, reliability, replication, stream processing, wide-area networks

1. INTRODUCTION

Borealis-R is a stream processing system for Internet-scale monitoring applications. In such applications, users want to monitor various events occurring around the world and make smart decisions in near real time. In practice, however, it is hard to accomplish correct and timely processing. For example, an overloaded server may stop sending data and a failed server may have lost data essential to processing. Furthermore, computer networks are vulnerable to link congestion and may experience outages.

[*]This work has been supported by the National Science Foundation under Grant No. IIS-0086057 and IIS-0325838.

In addition to distributed operation [2], Borealis-R has features that tackle the problems above. Similar to previous solutions for reliable stream processing [5, 4, 7, 3], it deploys, for each operator, k replicas on independent servers to tolerate up to $(k-1)$ simultaneous failures. In contrast to the previous solutions, however, Borealis-R makes multiple replicas send data to each downstream replica. As illustrated in Figure 1, this allows each replica to use whichever data arrives first. To further expedite processing, Borealis-R executes replicas without coordination, which causes non-deterministic replicas to run differently. Regardless of this flexible operation, Borealis-R guarantees that applications receive the results that a non-replicated execution would produce if the system was completely free from failures. Borealis-R also strives to achieve better performance and availability without increasing network cost.

In general, Borealis-R uses more computation and network resources than previous solutions in which only one of multiple replicas can feed downstream replicas. However, it also offers distinct advantages. First, it reduces latency because it always uses the fastest among multiple replicated data flows. Second, it is naturally resilient against server and network problems. In previous solutions, if a replica that feeds downstream replicas fails (or gets overloaded/disconnected), the processing stops until each downstream replica notices the problem after some delay and creates a new input connection from another functioning upstream replica. In contrast, Borealis-R stays operational without blocking to detect problems and switching between replicas.

2. MAIN FEATURES OF BOREALIS-R

To manage the system in a scalable fashion, Borealis-R groups servers into logical clusters, each of which comprises on the order of tens of servers. Each cluster consists of servers at diverse locations (rather than those only within a small area) in order to autonomously handle queries spanning distant stream sources and applications. As illustrated in Figure 1, Borealis-R guarantees fast and reliable processing by making multiple replicas feed each downstream replica. To achieve reliable processing with unstable stream sources, Borealis-R deploys entry points that keep input tuples from external sources until they safely arrive at the downstream replicas.

2.1 Replication Transparency

In Borealis-R, replication is transparent to users. In other words, each application always receives tuples as in the *ideal non-replicated execution* scenario where the system is com-

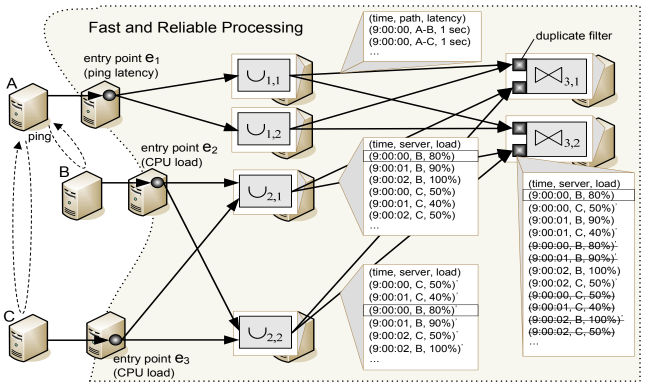

Figure 1: An Example of Replication – Servers B and C report their CPU usage via entry points e_2 and e_3, respectively. Replicas $\cup_{2,1}$ and $\cup_{2,2}$ commonly merge streams from e_2 and e_3 and feed $\bowtie_{3,2}$ in parallel. $\bowtie_{3,2}$ uses whichever tuple arrives first from $\cup_{2,1}$ and $\cup_{2,2}$, while ignoring duplicates (i.e., late tuples; see those lined-through).

pletely free from failures and delays. We call this property *replication transparency.*

A simple way to achieve replication transparency would be to identically execute peer replicas [3]. As will be shown in the demonstration, however, this approach introduces extra latencies. Therefore, Borealis-R runs replicas independently, while allowing them to process any available data. This causes multi-input operators to produce tuples in different orders. In Figure 1, replicas $\cup_{2,1}$ and $\cup_{2,2}$ produce outputs in different orders for this reason. Despite this, Borealis-R achieves replication transparency as follows:

1. *It merges stream replicas into a non-duplicate stream using a non-blocking duplicate filter* (cf. the second duplicate filter of $\bowtie_{3,2}$ in Figure 1). This is to prevent duplicate-sensitive operators, such as count aggregates, from producing incorrect results. Duplicate filters can handle disorder in stream replicas and multiple occurrences of the same tuple in a stream replica.

2. *It sorts disordered streams only when necessary.* Order-sensitive applications and operators (e.g., those with count-based windows) must receive inputs in the order of the ideal non-replicated execution. Borealis-R sorts streams only in such cases, because sorting streams introduces extra delays [6].

3. *It provides operators that always produce, from disordered input streams, the output tuples that would appear in the ideal non-replicated execution.* The output order is not important because downstream replicas can handle disorder. We call this property of operators *replica consistency* because it guarantees that operator replicas always produce consistent output streams from consistent input streams. We say that two streams are *consistent* if they contain the same tuples regardless of internal order.

In summary, Borealis-R achieves replication transparency by eliminating duplicates, minimally sorting data streams, and ensuring replica consistency. The details are presented in [6].

2.2 Management of Replicas

Borealis-R efficiently improves both performance and availability. For this, it initially creates replicas in a manner that minimally increases the network cost. To cope with changes in system conditions, it gradually discards the least useful replicas or adds new replicas (further details can be found in [6]):

Initial Deployment. As described early in this section, Borealis-R forms logical clusters to manage the system in a scalable way. Servers in the same cluster elect a coordinator that traces the network delays between them. Whenever the coordinator receives a query (i.e., a network of operators), it deploys replicas in a predefined number $k_{\max}$ of phases, each of which adds a new replica for each operator.

In each deployment phase, the coordinator first rules out servers that are too busy to run a new replica or likely to fall into the same network partition with a server that already runs a peer replica. Among the remaining candidate servers, the coordinator finds the server that will minimize the *network cost* of the input and output streams of the new replica. The network cost of each stream is defined as the product of the *data rate* and *latency* of the stream. This bandwidth-delay product is based on the observation that usually more network resources are used as more data stays in the network for a longer time. An optimal placement of replicas under this metric also tends to choose fast network links, thereby improving performance.

Garbage Collection. After deploying replicas, the coordinator periodically discards the least useful replicas until the overall network cost decreases to a predefined level θ. To preserve the minimum level of failure tolerance, however, it keeps at least $k_{\min}$ replicas for each operator. As a result, garbage-collection reduces the number of stream replicas from $k_{\max}^2$ (i.e., all possible connections between $k_{\max}$ upstream replicas and $k_{\max}$ downstream replicas) to a few times $k_{\min}$, while minimally degrading performance and availability.

To measure the utility (i.e., the impact on applications) of each stream replica, each duplicate filter gives, for each input tuple, different weights $\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \cdots$ to stream replicas according to how early these stream replicas deliver the tuple. For stream replicas $\{S\}_{i=1}^{k}$, the utility $u(S_i)$ of S_i is defined as $\frac{w(S_i)}{\sum_{j=1}^{k} w(S_j)} u(o)$, where $w(S)$ is the accumulated weight of stream S and $u(o)$ is the utility of the operator replica o that $\{S\}_{i=1}^{k}$ commonly feed. The utility of o, $u(o)$, is defined as $\sum_{S \in out(o)} u(S)$, where $out(o)$ denotes the output streams of o.

Adaptation. If an operator replica observes unusual delays in its input, it revives some garbage-collected input streams to reduce the input delay. If the current network cost is below the target cost θ, the coordinator also adds replicas to further reduce latency.

3. DEMONSTRATION DETAILS

The demonstration will stress the strengths of Borealis-R using a wide-area monitoring application. Specifically, the demonstration will show that (i) Borealis-R outperforms representative previous solutions in terms of latency, (ii) Borealis-R deploys replicas in a resource-efficient manner, and (iii) garbage-collecting replicas can significantly reduce the overall network cost while keeping latency low.

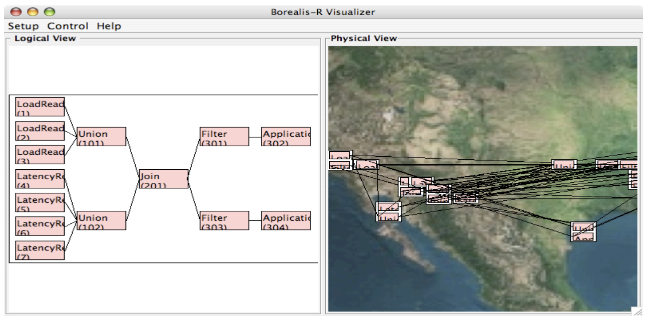

Figure 2: The Borealis-R Visualizer

3.1 Tools

The demonstration will proceed using the visualizer in Figure 2. The visualizer has the following views:

1. Logical View - The logical view displays the queries as graphs of operators and also allows us to edit the queries. We need to specify the actual locations of stream sources and applications. In contrast, intermediate operators are not assigned locations and Borealis-R automatically deploys their replicas.
2. Physical View - The physical view illustrates the deployment of replicas as well as various statistics such as the CPU utilization of each server, the data rate as well as the latency of each stream replica, and the latency at each operator replica. It also allows us to control the system in various ways, including adding more replicas, conducting garbage-collection, and killing a Borealis-R process.

3.2 Setup

The demonstration will show the operation of a logical cluster using a live wide-area monitoring application. We will choose tens of distant PlanetLab servers [1] that reliably communicate with each other. If the demonstration site has a network problem, we will run Borealis-R processes locally, while emulating network latencies based on a trace collected in advance.

The demonstration will assume the following scenario: "As illustrated in Figures 1 and 2, users want to monitor a subset of servers in the PlanetLab testbed. Each of these servers runs two periodic stream sources, one that reports the server's CPU load and the other that reports the recent latencies in the connections to other servers. The load readings are merged at $\cup_1$, whereas the latency readings are merged at $\cup_2$. Because users want to identify long latencies while determining whether such latencies are caused by busy remote servers or by the network itself, a Join operator correlates the load and latency readings. Then, the Join operator feeds operators that produce output streams according to users' various interests."

Given the query above, the coordinator will deploy replicas using the strategy described in Section 2.2. To successfully garbage-collect replicas later, the coordinator will continuously collect statistics, including network latencies between servers and the data rates of streams.

3.3 Comparison to Previous Techniques

This demonstration will compare Borealis-R with representatives of previous solutions [5, 4, 7, 3] during both non-failure and failure periods. We will first observe the runtime behavior of two modes. The Borealis-R mode will run multiple replicas without coordination, while sending their outputs to each downstream replica. The other mode will mimic previous solutions by allowing only one of many peer replicas to send data downstream. This will show that Borealis-R outperforms previous solutions during failure-free periods because it always uses the fastest among multiple replicated data flows.

We will also compare the two modes in terms of the failure masking capability. For this, we will randomly choose a Borealis-R process and kill it using the visualizer. This will show that Borealis-R keeps latency lower than previous solutions because it does not need to detect failures and switch between replicas.

3.4 Advantages of Replication

This demonstration will show how the replica deployment strategy in Section 2.2 can reduce latency while efficiently using network resources. To obtain a good initial, non-replicated deployment, we will first deploy $k_{\max} = 3$ replicas for each operator as described early in this section and then garbage-collect replicas with $k_{\min} = 1$ and $\theta = 0$. The reason behind this is that, when we deploy replicas for the first time, it is hard to find a good deployment since the data rate of each stream is not yet known. After finding a good non-replicated deployment, we will gradually add replicas and observe how the end-to-end latency varies. We will also contrast the above scenario with another one where replicas are deployed at arbitrary servers.

3.5 Garbage Collection

In this demonstration, we will show that garbage-collection can effectively reduce the overall network cost while keeping latencies low. We will also observe what types of replicas are usually less useful and thus discarded.

4. REFERENCES

[1] http://www.planet-lab.org.

[2] M. Balazinska, D. Abadi, Y. Ahmad, M. Cherniack, J.-H. Hwang, W. Lindner, A. Rasin, N. Tatbul, Y. Xing, and S. Zdonik. The design of the borealis stream processing engine. In *Proc. of the 2nd CIDR*, 2005.

[3] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker. Fault-tolerance in the borealis distributed stream processing system. In *Proc. of the 2005 ACM SIGMOD*, June 2005.

[4] J.-H. Hwang, , U. Çetintemel, and S. Zdonik. A cooperative, self-configuring high-availability solution for stream processing. In *Proc. of the 23th ICDE*, 2007.

[5] J.-H. Hwang, M. Balazinska, A. Rasin, U. Çetintemel, M. Stonebraker, and S. Zdonik. High-availability algorithms for distributed stream processing. In *Proc. of the 21th ICDE*, 2005.

[6] J.-H. Hwang, U. Çetintemel, and S. Zdonik. Fast and highly-available stream processing over wide area networks. In *Proc. of the 24th ICDE*, 2008.

[7] M. A. Shah, J. M. Hellerstein, and E. Brewer. Highly-available, fault-tolerant, parallel dataflows. In *Proc. of the 2004 ACM SIGMOD*, June 2004.