

Calder: Enabling Grid Access to Data Streams

Nithya Vijayakumar, Ying Liu, Beth Plale
Department of Computer Science, Indiana University
{nvijayak, yingliu, plale}@cs.indiana.edu

1. Introduction

As the world in which we live becomes increasingly sensed and monitored, data streams containing timely and continuously generated data, will be sought by more researchers with differing needs and skill levels. Bringing instrument data streams to the grid for broad community access is the challenge we address. The goal of Calder is to provide immediate, programmatic access to this data. Calder’s service-oriented architecture serves this need by establishing common interoperability guidelines that enable discovery and communication between a scientist’s application and a data resource.

The motivation for our work is a dynamic, adaptive meteorology forecasting system. In a traditional forecasting process, recent data obtained from observational sources (*i.e.*, satellite, buoys) are assimilated into a single 3D grid and passed as input to a forecast model such as WRF or ADAS. In future, researchers will want to examine the results of forecast runs for regions of greatest uncertainty and direct data gathering devices, such as local radars, to that region to bring in localized data. We view this scenario as one of connecting an application, in this case an ensemble forecast, to one or more observational instrument data streams on demand. We view a fitting solution to this problem in a synchronization buffer between the data streams and the consumer and simple select-project-join queries for fusing and filtering streams before they arrive at the buffer.

In a *continuous query* model, the queries are deployed onto computational nodes in a network and execute continuously, over the streaming data [5, 2, 1]. Calder extends OGSA-DAI [3] and dQUOB [5]. Calder’s service interface enables programmatic access - the virtual piping of stream directly into a processing application. The model of bringing data streams to the grid as a single, coherent data resource was first introduced in [4]. The prototype Calder system implements this model. Our contribution is a feasibility analysis of this continuous query grid service.

In order to establish feasibility of the grid-based continuous query solution implemented by Calder, we experimentally evaluate the overhead of deploying a new query to

the computational network (*Query deployment time*), performance of a single computational node under a growing workload of queries (*Resultset time interval*), and performance of the rowset service under an increasing load of user requests (*User turnaround time*).

2. System Overview

The five main architectural components of Calder, shown in Figure 1, are a grid data service, a distributed query planner service, a persistent rowset data service, a set of managed computational nodes, and a distributed query processing server. The **Grid data service** (GDS) is a transient grid service instantiated on a per user basis. The Calder GDS extends the OGSA-DAI v4 grid data service [3] which extends Globus. A user accesses the Calder system by creating a GDS and passing it an XML document containing the continuous query. The GDS connects to the persistent distributed query planner and passes it the raw SQL query. The **distributed query planner** is a separate persistent service that uses query reuse techniques and optimizes the query for distribution across the computational network. The distribution decision for a query is passed to the **query processing server**, which deploys the query to one or more computational nodes. Running at each **computational node** in the network is a runtime (a “*quoblet*”) that dynamically accepts queries as scripts and deploys them as compiled code. Details of quoblet functionality is given in [5].

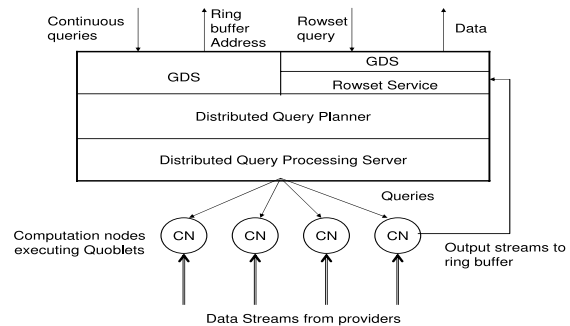


Figure 1. Calder Architecture

GDS Setup	Planning	Compile	Deployment	RB Setup
2951 ms	15 ms	24 ms	2 ms	24 ms

Table 1. Query Deployment Time

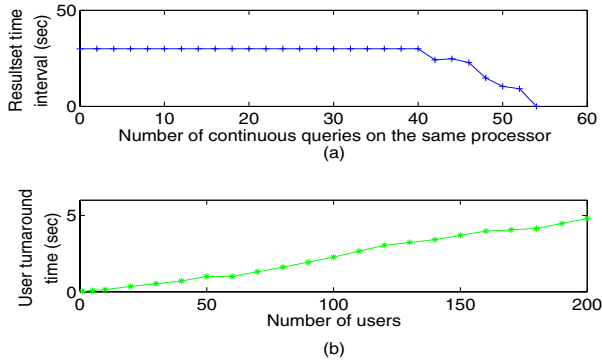


Figure 2. (a) Resultset time interval. (b) User turnaround time.

The rowset service is a buffer between timely streams and programs that access this data. It maintains a ring buffer (RB) of event data, one ring buffer per active query in the network. Results are obtained by means of time-based requests to the rowset service. A program can request a single event based on a timestamp, a range of events based on a time range, or can request the events that have arrived since the last request on a specific ring buffer. Currently a gridFTP URL is returned to the user for retrieval of the zipped netCDF files themselves. We are working on a solution to streaming these files to the user.

3. Experimental Evaluation

In the experimental setup of the system, the grid data service (GDS), rowset service and the query processing server reside on a dual processor Dell 2.8 GHz Precision workstation, 2 GB memory, running RHEL. The Planner service runs on a Sun UltraSPARC-IIe 502MHz processor, 4GB memory running SunOS 5.8. The computational node runs on a dual processor Xeon Intel 2.8GHz, 2GB memory, running Linux RedHat 8.0. The machines are interconnected through 1 Gbps switched Ethernet LAN.

Query deployment time is the time from which a new query is passed to the Calder’s GDS, through the query planner service and query processing server, to instantiation on a node in the computational network. Included in the procedure is the allocation of a new ring buffer (RB) in the persistent rowset service. Table 1 lists the distribution of *query deployment time*. The results are as expected. GDS creation time is a one time cost per user. We are focusing on removing this overhead through a persistent GDS.

Resultset time interval is the time interval of the returned

result set. We assume the user issues a “getMostRecent” command to the rowset service which returns all events since the last call. The scalability of a single computational node with respect to the number of active queries is captured in the size of the *resultset time interval* in Figure 2 (a). As the computational node falls behind, the user receives fewer events than when the the node is running in a steady state. Our measurement of 40 queries is based on a synthetic workload with an arrival rate of 50 events/second. From our understanding of meteorology forecasting, we estimate that the average user requires an aggregate generation rate of less than 1 event per second. This enables roughly 200 queries to be supported per computational node.

User turnaround time is the user perceived turnaround time for retrieving results from the rowset service. We examine the impact of additional users at the persistent rowset data service on the turnaround time experienced by the user. From Figure 2 (a), we fix the number of queries at 40. *User turnaround time*, plotted in Figure 2 (b), shows a steady increase under an increasing number of ‘users’. Event meta-data is stored in an in-memory ring. When a rowset request resolves into a sequence of multiple events on the ring buffer, the rowset service merges the file data into a single file, and returns the URL of that file to the user. This high I/O load is responsible for the steady increase in turnaround time. This inefficiency is being removed from the system. We expect performance to improve significantly as a result.

4. Conclusion and Future Work

This paper presents a feasibility study of a grid-based continuous query solution to access data streams. The results presented in this study are mixed, however. We are migrating to a netCDF-based streaming model to measure system behavior in a setting that more accurately reflects a real use scenario. We are also working on enabling approximate query processing support to Calder to deal with sudden drop offs and changes in stream rates.

References

- [1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *ACM, PODS*, 2002.
- [2] D. J. Abadi, Y. Ahmad et al. The Design of the Borealis Stream Processing Engine. In *CIDR*, 2005.
- [3] M. Antonioletti, M. Atkinson et al. Design and implementation of grid database services in OGSA-DAI. *Concurrency and Computation: Practice and Experience*, 17, 2005.
- [4] B. Plale. Using global snapshots to access data streams in the grid. In *Lecture Notes in Computer Science Series*, 2004.
- [5] B. Plale and K. Schwan. Dynamic querying of streaming data with the dQUOB system. *IEEE Transactions in Parallel and Distributed Systems*, 14, 2003.