An Overview of the Granules Runtime for Cloud Computing

Shrideep Pallickara, Jaliya Ekanayake and Geoffrey Fox
Community Grids Laboratory
Indiana University
Bloomington, IN 47401, USA
{spallick, jekanaya, gcf}@indiana.edu

***Abstract* — In this paper we present a short introduction to the Granules system, which is a lightweight streaming-based runtime for cloud computing. This paper provides a summary of the capabilities supported by the runtime.**

Keywords: cloud computing, streaming systems, map-reduce, runtime

I. INTRODUCTION

Cloud computing enables applications to harness capabilities, both computing and storage, that are available in computational clouds that comprise thousands of computers. In the past year, cloud computing environments have gained significant traction. Salesforce allows clients to purchase entire solutions that are hosted on their cloud. In some cases, such as Google, the generated responses are the result of processing that was performed in the cloud. Amazon, on the other hand, allows direct access to its cloud: users can allocate and deallocate entire virtual machines. Typically, most of these cloud computing solutions have been used for commercial applications such as processing web documents, videos, images, payroll, and so forth.

The sheer scale of the computing and storage capabilities available in datacenters make cloud computing particularly well-suited for scientific applications. These scientific applications can be compute-intensive, data-intensive or both: in each case the processing needs are significant enough to mandate concurrent processing.

It can be argued that cloud computing is a natural evolution of the grid computing paradigm wherein the services are hosted on a compute cloud that can comprise a much larger set of machines.

II. GRANULES

The Granules project is a lightweight streaming-based runtime for cloud computing. Granules orchestrates the concurrent execution of applications on multiple machines. The runtime manages an application's execution through various stages of its lifecycle: deployment, initialization, execution and termination. Granules uses NaradaBrokering [1] as the content distribution network for disseminating streams. At each computational resource, Granules manages the concurrent execution of multiple application instances. Fig. 1 depicts the components that comprise Granules.

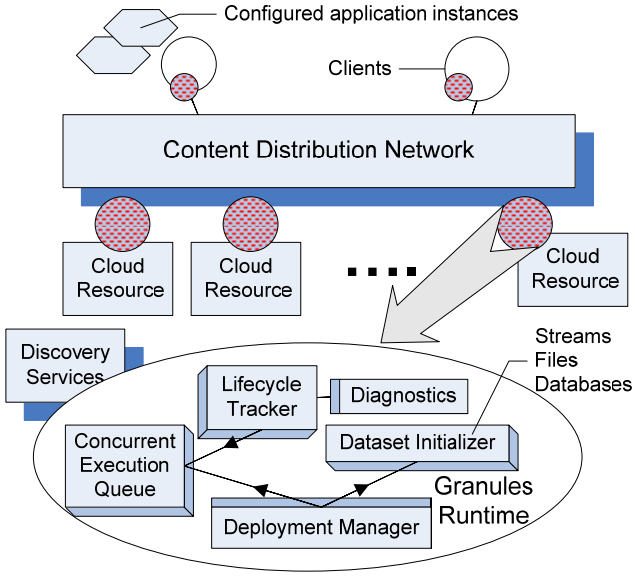

Figure 1. Figure 1: Overview of the Granules runtime

One of the characteristics of cloud computing is that the resources are dynamic. If an application programmer is forced to keep track of all the available computational resources, the concomitant increase in the application's complexity would be significant. Granules obviates the need for applications to track resource availability. Instead, applications delegate the responsibility of discovering and harnessing computational resources to Granules. Depending on the requirements, Granules can deploy application instances on one or more computational resources.

Besides incorporating the processing functionality, application instances need to specify the datasets (or sources thereof) that will be processed during execution. An application may generate its own inputs or operate on external datasets such as files, streams and databases. The application can also generate outputs in a variety of formats.

Granules performs two key steps during the deployment of the application instance. First, it initializes the state of the application instance based on the specified initialization directives. Second, it initializes the datasets that the application operates on. Dataset initializations involve subscribing to the appropriate data streams, configuring

access to files on the networked file system, or setting up connections to databases.

Granules also allows application instances to specify an execution profile that will govern their lifetime and scheduling strategy. In addition to the traditional exactly-once semantics for execution, application instances can be executed a specified number of times, at regular intervals, or a combination thereof. Application instances can also be stay-alive with the execution being scheduled periodically, whenever additional data is available, or until the application instance asserts that the processing is complete.

After every scheduled execution, Granules checks to see if the application instance is ready for termination based on the specified execution profile. If it is determined that an application instance is ready for termination, Granules removes it from the concurrent execution queue and reclaims any allocated resources. Reclaiming involves unsubscribing from any of the streams that the application instance was consuming and clearing any file locks that were established. Components that register for diagnostic messages related to task executions are notified about any lifecycle transitions.

III. STREAMING MAPREDUCE

The map-reduce [2] programming model (depicted in Fig. 2) facilitates the concurrent processing of large datasets. Here, large datasets are split into smaller more manageable sizes which are then processed by multiple *map* instances. The results produced by individual map functions are then sent to *reducers*, which collate these partial results to produce the final output. A clear benefit of such concurrent processing is a speed-up that is proportional to the number of computational resources. The most popular implementation of the MapReduce framework is Hadoop [3], which relies on the Hadoop Distributed File System.

In Granules, we have incorporated support for a stream-based implementation of the MapReduce framework. The streaming implementation of MapReduce in Granules has two distinct advantages over file-based implementations. First, intermediate results are shared between the computational units using streaming, instead of files that require disk IO, which can add considerable overheads. The second advantage stems from the fact that results can be streamed to the reducers as they become available instead of having to wait for the entire processing to be complete. This feature is particularly relevant in situations where there is often a need to get as many results as possible within a fixed amount of time. We have demonstrated [4] the suitability of streaming-based MapReduce for certain classes of data intensive scientific applications.

Typical MapReduce stages look like a directed acyclic graph with the MapReduce execution progressing in monotonically increasing stages. Besides the basic support for MapReduce, we have incorporated support for variants of the MapReduce framework that are particularly suitable for scientific applications. This includes support for iterative and recursive implementations of the framework. We are incorporating support for cycles wherein the outputs of the reduction stage could themselves be inputs to the preceding map stages. This feature is particularly useful in clustering

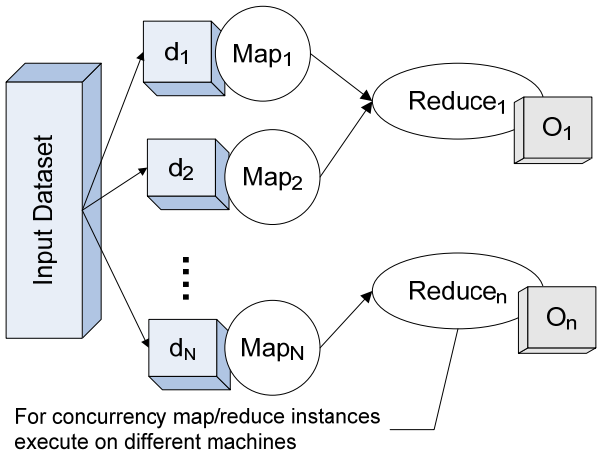

Figure 2. Figure 2: The MapReduce programming model

algorithms where the computed (reduced) results need to be refined, to meet a targeted error rate, by the map stages.

IV. RELATED WORK

In this section we present a brief overview of the some of the related work in this area. Dryad [5] is a distributed execution engine for coarse grain data parallel applications. Dryad combines the MapReduce programming style with dataflow graphs, that are directed and acyclic, to orchestrate computational tasks. Phoenix [6] is an implementation of MapReduce for multi-core and multiprocessor systems. Disco [7] is an open source MapReduce runtime developed using the Erlang functional programming language. Similar to the Hadoop architecture, Disco stores the intermediate results in local files and accesses them using HTTP connections from the appropriate reduce tasks.

ACKNOWLEDGMENT

This research is supported by grants from the National Science Foundation's Division of Earth Sciences (EAR-0446610) and the Division of Information and Intelligent Systems (IIS-0536947).

REFERENCES

[1] S Pallickara and G Fox. "NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids". Proceedings of the ACM/IFIP/USENIX International Middleware Conference Middleware-2003. pp 41-61.

[2] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," ACM Commun., vol. 51, Jan. 2008, pp. 107-113.

[3] Apache Hadoop, http://hadoop.apache.org/core/

[4] J. Ekanayake, S Pallickara and G. Fox. "Map-Reduce for Scientific Applications". Proceedings of the *IEEE International Conference on e-Science*. Indianapolis. 2008.

[5] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," European Conference on Computer Systems , March 2007.

[6] Disco project, http://discoproject.org/

[7] C. Ranger, R. Raghuraman, A. Penmetsa, G. R. Bradski, and C. Kozyrakis. "Evaluating MapReduce for Multi-core and Multiprocessor Systems," Proc. International Symposium on High-Performance Computer Architecture (HPCA), 2007, pp. 13-24.