Multi-stage Distributed Computing with BOINC
Vance Zuo

Advisor: Holly Rushmeier

SUMMARY

In recent years, “big data” has spurred the development of distributed computing frameworks. One popular framework for processing large datasets has been BOINC, an open-source framework designed for volunteer computing but easily adapted to closed cluster computing applications. BOINC is useful in particular for the graphics and image processing at the Yale Graphics Group, offering a great deal of flexibility compared to alternatives. However, BOINC projects are currently not well-suited for multi-stage pipelines, especially those that involve integrating outputs from one stage into inputs for another. I propose creating a layer above BOINC for defining and automatically executing such pipelines. This has immediate applications for existing projects in the Yale Graphics Group, but should also be flexible enough to be adapted towards other uses.

BACKGROUND

BOINC (Berkeley Open Infrastructure for Network Computing) was created in 2002 to manage SETI@home, a volunteer computing project that processes radio data for potential evidence of extraterrestrial intelligence. Since then, its use has expanded—as of September 2015, there were 56 active public projects and over 3 million contributors.¹

Internally, BOINC follows a master-worker model; a server sends tasks to clients, which execute the tasks and then upload the results back to the server. Because computing resources may come from potentially untrustworthy volunteers, it emphasizes security and accountability—for example, tasks can be sent to multiple machines, and their results compared with each other to verify correctness.

¹ http://boincstats.com/en/stats/projectStatsInfo
It is also designed to handle heterogeneous computing resources, since available computing power and time can differ widely across clients.

These features give BOINC distinct advantages for the Yale Graphics Group's graphics and image processing projects compared to other frameworks, such as Hadoop MapReduce and Spark. These alternatives tend to be more effective for low latency applications, but at the cost of flexibility. The use cases where they excel tend to have:

- Homogenous nodes; i.e., similar if not identical computing power
- Huge data files, ideally at least gigabytes in size
- Computations that fit a MapReduce programming model (which allow a task to chopped up into many pieces to distribute)

Meanwhile, the Graphics Group's projects have:

- Heterogeneous nodes, ranging from commodity PCs to supercomputer nodes
- Image file sizes in megabytes
- Complex, monolithic computations that are difficult to translate into MapReduce

Workarounds to these problems do exist in the alternative frameworks, such as concatenating images together to create the large data files preferred by Hadoop and Spark, but they introduce a lot complexity to solve problems that for BOINC don't exist. Thus, BOINC has the preferred framework for distributing the Graphics Group's work.

That said, there is room for improvement. BOINC currently has few tools for managing projects that contain multiple related computations—for instance, a pipeline that consists of extracting texture information from images, then computing similarities between these results. To execute a multi-stage pipeline like this would require manually constructing inputs from outputs of a project and submitting them to another project—time-consuming, especially considering the size of the data sets.
My goal is to create a system for managing multiple BOINC projects working together in a pipeline with the following features:

- Web submission interface
- Database for accessing inputs and outputs of all projects
- Method for defining job prerequisites for stages in a pipeline
- Automatic creation and execution of jobs in stage when prerequisites are met

**SPECIFICATION**

The system will make use of the following technologies:

- MySQL
- PHP
- BOINC

A MySQL database will function as a central location containing resources and state from all the BOINC projects in a pipeline. The database will also have timestamps for measuring performance. Its schema will capture the following concepts:

- **Pipeline** – represents a set of inputs and all jobs and resources originating from them
- **Job** – BOINC task associated with a particular a pipeline and certain inputs
- **Resource** – information on data belonging to a pipeline, including inputs and outputs for jobs
- **Job Dependency** – a prerequisite for a job

PHP scripts will manage and update pipeline state, including:

- Install/uninstalling the database
- Generating pipeline and resource entries in the database for a given set of inputs
Managing jobs, e.g., submitting tasks to BOINC for jobs whose prerequisites are met, and updating resource entries for outputs of finished jobs.

PHP scripts will also be used to define job dependency logic—that is, what new jobs to generate when one or more jobs are completed.

The system will also have basic security features, e.g., preventing SQL injections and requiring tokens to access resources.

**DELIVERABLES**

By the end of the semester, I intend to have:

- All of the scripts described in the specification
- Demos of the system with:
  - A simple image processing application
  - A graphics texture processing application in the Graphics Group
  - A materials capture application in the Graphics Group
- A written report describing in greater detail the components of the system—the database schema, PHP modules, etc.—as well as its performance (i.e., latencies) compared to a single machine setup. It will also discuss successes and challenges using the system with the demo applications.