CS 578 – Software Architectures
Fall 2014
Homework Assignment #1
Due: Wednesday, September 24, 2014
– see course website for submission details –

The Berkeley Open Infrastructure for Network Computing (BOINC) Case Study, provided as an appendix to this assignment, presents an initial high-level (“Level 1”) architectural breakdown for the system. The overall BOINC system comprises several components, one of which is the BOINC task server.

1. The lectures and readings have begun discussing the architectural design process, including the selection of architectural styles. Not all styles are applicable to all systems and any choice of style will involve making trades on various system attributes. Pick any two styles that you have read about and design two different architectures for the BOINC task server, one that adheres to each style. In order to apply a style, you will need to create a detailed architectural breakdown (a “Level 2” architectural breakdown) for the task server. In other words, “expand” the task server box shown in Figure 4 in the Case Study into an architecture. Make sure to show all the connectors inside the task server architecture as well as those that interconnect the task server to the other parts of the system. Moreover, you should graphically distinguish different types of the connectors used in your design. Also, there is no such thing as the “correct” or “optimal” architecture. However, as a granularity guideline, your decomposition of the task server should consist of no less than 15 distinct components.¹ You are not required to select a style from the course text book, but you must let us know what style you are attempting to apply and provide a reference if the style is not one from the course text. Submit one diagram for each architecture you design.

2. Give a brief rationale for your architecture: why did you select the styles that you did? Weigh the pros and cons of each architectural style. We will not grade you based on how accurately you apply each style as much as your rationale for selecting a particular style and understanding its limitations. Please limit your answer to 2 paragraphs.

3. Compare each of your architectures: give one example of a system property/requirement described in the Case Study that is addressed in a superior manner by one of your architectures. Be sure not only to name the property/requirement, but also to explain how each architecture addresses the property/requirement and give rationale for why you think one architecture is superior to the other. Please limit your answer to 2 paragraphs.

4. Since BOINC is a very large system with many different, possibly conflicting, requirements, your architecture may only directly address a subset of those requirements. To demonstrate this, for one of your architectures, select two of the key architectural challenges and requirements (listed in bulleted items in the last two pages of the Case Study) and argue/discuss how your architecture DOES NOT support them in an acceptable fashion. Please limit your answer to 1 paragraph.

5. Add a capability to both your architectures to integrate one or more cloud(s) into the BOINC computation. The project operator must be able to vary the amount of cloud computation power involved in the BOINC computation based on the active volunteer-participant numbers. How will you change each architecture in response to this new requirement? Which architecture can be changed more easily? Why? Please limit your answer to 2 paragraphs and, if you find it necessary, one diagram for each of your architectures.

¹ This is only a guideline. There is nothing magical about this number, nor do we have a specific solution in mind.
Berkeley Open Infrastructure for Network Computing

(A case study)

Note: this is a simplified and generalized description of a real system

Introduction

The Berkeley Open Infrastructure for Network Computing (BOINC) is an open-source software for volunteer computing and grid computing. The primary function of the system is to make it possible for researchers to tap into the enormous processing power of personal computers around the world.

BOINC was originally developed to support the SETI@home project (http://setiathome.ssl.berkeley.edu), an experiment that analyzes radio telescope data from outside the Earth as an attempt to find extraterrestrial intelligence. The challenge was that the analysis of radio telescope signals required enormous amount of computation resource as the received signals consisted of much noise and man-made signals. The more computation resource involved, the wider frequency range coverage with more sensitivity. Instead of adopting high-cost supercomputers for the computation, a high performance distributed computing platform, BOINC, was developed to take advantage of the idling cycles of personal computers of people around the world who wanted to participate in the project.

Overtime, BOINC has become useful as a platform not just for searching for extraterrestrial intelligence but also for other distributed applications in areas as diverse as mathematics, medicine, molecular biology, climatology, environmental science, and astrophysics. It has about 320,121 active participants and 512,197 active computers (hosts) worldwide processing on average 6.6 petaFLOPS as of July 23, 2014.

The distributed nature and the gigantic scale of BOINC bring up several design challenges. For example, it must be able to manage a large pool of participating nodes (personal computers) remotely via computer networks. There could also be performance, security, and/or scalability concerns when designing its architecture. This is discussed in more detail later.

Volunteer Computing: How It Works

Participant’s Perspective

Volunteer computing performs computation using participants’ computation resource. Figure 1 depicts the participant’s perspective of a volunteer computing system. The following steps are indefinitely iterated:

1. A participating node (denoted as “Your PC” in the figure) receives a set of instructions,
2. The node downloads the executable applications and the input files for computation,
3. The node performs the computation and produces output files,
4. The node uploads the output files to the server, and
5. The node reports the list of completed tasks to receive reward and to adjust the size of future jobs.

![Figure 1: General Depiction of Volunteer Computing from the Participant’s Perspective](image-url)

Refer to [http://setiathome.berkeley.edu/sah_about.php](http://setiathome.berkeley.edu/sah_about.php) for more information.
Project’s Perspective

Volunteer computing provides an opportunity to projects that need large computation power and/or storage via collecting the resources from volunteers who wish to participate in the project. While the specifics of how a computation is performed differs system to system, but the high-level computational model (i.e., how the computation is divided, performed, and assimilated) is identical. Figure 2 depicts a volunteer computing system model in project’s perspective. The task server receives a computation as its input. The computation is divided into multiple computation tasks. A task is a part of the computation that is independent from the other tasks. Each task is duplicated into redundant jobs, which are the ones transferred in the second step of Figure 1. With redundancy, each task is executed as several identical jobs on distinct nodes. The created jobs are assigned to a randomly selected participant node via the job queue. When the nodes report back the results of the assigned jobs (step 4 of Figure 1), the task server compares and accepts the results. New jobs could be created based on the results. This process is iterated until no more jobs are left in the job queue.

![Figure 2: General Depiction of Volunteer Computing from the Project’s Perspective](image)

System Description

Overall System Description

BOINC has the general goal to nurture the public resource computing. It has the following specific goals:

- Reducing the barriers of entry to public-resource computing
- Sharing resources among autonomous projects
- Supporting diverse applications
- Rewarding participants

The Level 1 architecture (depicted in Figure 1) of BOINC is in the Client-Server architectural style. The architectural breakdowns of the BOINC client and server are in the following subsections.
BOINC Client

BOINC client consists of four major components and communicates with the components in BOINC server. Figure 3 depicts the BOINC client. The list of components is as following:

- The *schedulers and data servers* are installed on computers owned and managed by the projects to which the volunteers donate time of their computers.
- The *core client* communicates with the BOINC servers via the HTTP communications protocol to get and report work. The core client also runs and controls applications.
- *Applications* are the programs that do scientific computing. Several of them may run at the same time on a computer with more than one CPU.
- The *GUI* provides a graphical interface that lets the volunteers control the core client – for example, by telling it to suspend and resume applications. The GUI communicates with the core client via a TCP connection. Normally this is a local connection; however, it's possible to control a core client remotely.
- The *screensaver* runs when the participants are away from the computer. It communicates with the core client via local TCP, instructing it to tell one of the applications to generate screensaver graphics.

![Figure 3: High-Level Architecture of BOINC Client](image)

BOINC Server

BOINC server, including the task server, is depicted in Figure 4. It has three major components:

- *Web interfaces* for account and team management, message boards, and other features.
- *Task server* that creates tasks, dispatches them to clients, and processes returned tasks.
- *Data server* that downloads input files and executables, and that uploads output files.

![Figure 4: High-Level Architecture of BOINC server](image)
**BOINC Task Server Breakdown**

The BOINC task server must have at least the following eight major components:

- The *work generator* creates new jobs and their input files. For example, the SETI@home work generator reads digital tapes containing data from a radio telescope, divides this data into files, and creates jobs in the BOINC database. The work generator sleeps if the number of unsent jobs exceeds a threshold, limiting the amount of disk storage needed for input files.

- The *scheduler* handles requests from BOINC clients. Each request includes a description of the host, a list of completed jobs, and a request for additional work, expressed in terms of the time the work should take to complete. The reply includes a list of jobs and their corresponding tasks. Handling a request involves a number of database operations: reading and updating records for the user account and team, the host, and the various jobs and instances.

- The *feeder* streamlines the scheduler’s database access. It maintains a shared-memory segment containing (1) database tables such as applications, platforms, and application versions, and (2) a cache of unsent jobs. The scheduler finds jobs that can be sent to a particular client by scanning this memory segment.

- The *transitioner* examines tasks for which a state change has occurred (e.g., a completed job has been reported). Depending on the situation, it may generate new jobs, flag the task as having a permanent error, or trigger validation or assimilation of the task.

- The *validator* compares the instances of a job and selects a canonical instance representing the correct output. It determines the credit granted to users and hosts that return the correct output, and updates those database records.

- The *assimilator* handles tasks that are “completed”: i.e., that have a canonical instance or for which a permanent error has occurred. Handling a successfully completed task might involve writing outputs to an application database or archiving the output files.

- The *file deleter* deletes input and output files that are no longer needed.

- The *database purger* removes tasks and job database entries that are no longer needed, first writing them to XML log files. This bounds the size of these tables, so that they act as a working set rather than an archive. This allows database management operations (such as backups and schema changes) to be done quickly.

**Key Architectural Challenges**

Here is a partial list of architectural challenges of BOINC, the ones that must be considered for designing the BOINC task server. Note that the challenges are not completely independent; hence there could be some repetition in the list.

- BOINC must be resilient to erroneous computation results from nodes.
  - There could be malfunctioning computers or malicious nodes.
  - *Redundant computing* could be implemented to overcome this challenge. Replicate tasks into multiple jobs, distributed the jobs to random nodes, compare and accept a *canonical result* when the computation results are received from the nodes.

- BOINC server must be able to handle a large number BOINC clients.
  - It must be able to handle a large number of network connections with BOINC clients.

---

1 This is called the Traditional Redundancy. Refer to [4] for more information.
o The BOINC task server has to be able to process a large number of simultaneous jobs performed by BOINC clients.

• BOINC server must be able to credit rewards to participants for successful computation results (e.g., the results that have been accepted as the canonical result).
  o The reward could be in the form of points and ranking between the participants.
  o The rewarding system must be able to vary the amount of credit based on the amount of resource (computation, storage, and network transfer) the participants provided.
  o The rewarding system must be able to work across different projects and applications.

• BOINC server must be able to adjust the size of jobs it assigns to BOINC clients based on the clients’ previous performance on the jobs that had been assigned to those clients.

• BOINC server must be available nearly 24/7.

References

This appendix is a summary of the following list of articles: