UNIVERSITY OF SAN FRANCISCO | College of Arts and Sciences

# <Master of Science/Computer Science/Major>

**ASSESSMENT REPORT**
**ACADEMIC YEAR 2017 – 2018**

## I. LOGISTICS & PROGRAM LEARNING OUTCOMES

1. **Please indicate the name and email of the program contact person to whom feedback should be sent (usually Chair, Program Director, or Faculty Assessment Coordinator).**

   EJ Jung, ejung2@usfca.edu, Faculty Assessment Coordinator of CS dept.
   Dave Wolber, wolberd@usfca.edu, Chairperson of CS dept.

2. **Were any changes made to the program mission statement since the last assessment cycle in October 2017? Kindly state "Yes" or "No." Please provide the current mission statement below. If you are submitting an aggregate report, please provide the current mission statements of both the major and the minor program.**

   No changes were made.

   The mission of the MS in Computer Science graduate program is:

   To provide students a strong theoretical background in computer science and deep technical programming skills by focusing on one-on-one student interaction and fostering the unique capabilities of each student.

   Our mission statement coincides with the university mission to give students the knowledge and skills needed to succeed as professionals, and we are sensitive to the needs of our extremely diverse student population.

3. **Were any changes made to the program learning outcomes (PLOs) since the last assessment cycle in October 2017? Kindly state "Yes" or "No." Please provide the current PLOs below. If you are submitting an aggregate report, please provide the current PLOs for both the major and the minor programs.**

   No changes were made.

Students who graduate with a MS in Computer Science will be able to:

Demonstrate advanced knowledge in a breadth of topics in computer science, including theory, systems, and development.

Demonstrate mastery in at least one area of specialization in computer science.

Demonstrate ability to independently solve advanced problems in academia or industry.

Demonstrate ability to learn, use, and adapt emerging developments in the state-of-the-art in computer science.

**4. Which particular Program Learning Outcome(s) did you assess for the academic year 2017-2018?**

Demonstrate mastery in at least one area of specialization in computer science.

## II. METHODOLOGY

**5. Describe the methodology that you used to assess the PLO(s).**

We chose to evaluate this learning outcome in the Special Topics course in Big Data taught in Fall 2017. Mastery in Big Data is defined by achieving the learning outcomes listed below per syllabus: Those students who successfully pass this class should be able to do the following:

- Leverage and design big data frameworks and distributed systems to carry out analysis and gain insight (assessed by projects)
- Preprocess and prepare data for machine learning, visualization, and summarization (assessed by projects)
- Critique research papers from the field (assessed by scientific paper reviews, in-class discussions)

We used Direct Methods, evaluating students project submissions, in-class discussions, and scientific paper review submissions. For the rubric, we used Project 1 as a representative assessment to provide the detailed rubric because it involves both design and implementation of a distributed system, and touches on some parts of analysis/preprocessing/data preparation (including dividing data up across processing units). Students were required to design their project beforehand using UML, diagrams, and pseudocode to produce a design document. The implementation of the project followed, and students were allowed to amend and update their design during development. In particular, students were required to design the system for a specific type of data (e.g., atmospheric data for predicting weather). At the end of the projects, students were

also required to write a retrospective document evaluating what went well and what could be improved were they to take on a similar project in the future. The rubric of project 1 is attached in the Appendix.

## III. RESULTS & MAJOR FINDINGS

**6. What are the major takeaways from your assessment exercise?**

**This section is for you to highlight the results of the exercise. Pertinent information here would include:**

**a. how well students mastered the outcome at the level they were intended to,**

**b. any trends noticed over the past few assessment cycles, and**

**c. the levels at which students mastered the outcome based on the rubric used.**

To assess mastery, we split the students into four groups:

1. "complete mastery of the outcome"
2. "mastered the outcome in most parts",
3. "mastered some parts of the outcome"
4. "did not master the outcome at the level intended."

The table below lists how many students fell into each category for each assignment group:

| Description | Complete Mastery | Mastery in Most parts | Mastery in Some Parts | Did not master at the intended level |
|---|---|---|---|---|
| Discussions | 29 | 2 | 0 | 0 |
| Project 1 | 21 | 9 | 1 | 0 |
| Project 2 | 18 | 8 | 4 | 0 |
| Project 3 | 24 | 5 | 1 | 0 |
| Review 1 | 10 | 10 | 5 | 4 |
| Review 2 | 16 | 5 | 8 | 2 |
| Review 3 | 15 | 7 | 6 | 2 |
| Review 4 | 21 | 8 | 2 | 0 |
| Review 5 | 24 | 4 | 1 | 2 |
| Review 6 | 19 | 12 | 0 | 0 |
| Review 7 | 23 | 8 | 0 | 0 |
| Review 8 | 25 | 6 | 0 | 0 |
| Final Grade | 24 | 5 | 2 | 0 |

Two students had some attendance issues that led to less than complete mastery of the discussion work. Research paper reviews trended upward over the course of the semester: early on, students tended to only summarize the work rather than critiquing it. Towards the end of the semester, they were better at identifying weaknesses in the work and suggesting alternative approaches. For projects, students were allowed to correct their work and resubmit for half credit back. This took some extra time but improved the number of students reaching complete mastery.

This is the first year that we evaluated this particular learning outcome, so we have not found any trends yet.

## IV. CLOSING THE LOOP

7. **Based on your results, what changes/modifications are you planning in order to achieve the desired level of mastery in the assessed learning outcome? This section could also address more long-term planning that your department/program is considering and does not require that any changes need to be implemented in the next academic year itself.**

   We are pleased that 94% of the students mastered the learning outcome in most parts. We noticed that the students who had attendance issues showed less than ideal performance and mastered only some parts of the outcome, and we will be more proactive on reaching out to students who have attendance issues in the early weeks of semester. We have a dedicated team of graduate director and graduate program manager who will support this early intervention.

8. **What were the most important suggestions/feedback from the FDCD on your last assessment report (for academic year 2016-2017, submitted in October 2017)? How did you incorporate or address the suggestion(s) in this report?**

   N/A

## ADDITIONAL MATERIALS

**(Any rubrics used for assessment, relevant tables, charts and figures should be included here)**

# CS686 Project 1: Distributed File System (v 1.4)

**Due: October 18**

In this project, you will build your own distributed file system (DFS) based on the technologies we've studied from Amazon, Google, and others. Your DFS will support multiple *storage nodes* responsible for managing data. Key features include:

- **Parallel retrievals**: large files will be split into multiple *chunks*. Client applications retrieve these chunks in parallel.
- **Interoperability**: the DFS will use Google Protocol Buffers to serialize messages. *Do not use Java serialization*. This allows other applications to easily implement your wire format.
- **Fault tolerance**: your system must be able to detect and withstand two concurrent storage node failures and continue operating normally. It will also be able to recover corrupted files.

Your implementation must be done in Java, and we will test it using the *bass* cluster here in the CS department. Communication between components must be implemented via sockets (*not* RMI or similar technologies) and you may not use any external libraries. The Java Development Kit has everything you need to complete this assignment.

Since this is a graduate-level class, you have leeway on how you design and implement your system. However, you should be able to explain your design decisions. Additionally, you must include the following components:

- Controller
- Storage Node
- Client

## Version Control

To set up your submission repository on GitHub, visit:
https://classroom.github.com/a/Yoknj2ce

In the spirit of versioning, I will update the version number at the top of this document every time a change is made and list any changes in the changelog below.

## Controller

The Controller is responsible for managing resources in the system, somewhat like an HDFS NameNode. When a new storage node joins your DFS, the first thing it does is contact the Controller. The Controller manages a few data structures:

- A list of active nodes
- A list of files
- For each file, a list of its chunks and where they are located

When clients wish to store a new file, they will send a *storage request* to the controller, and it will reply with a list of destination storage nodes to send the chunks to. The Controller itself should **never** see any of the actual files, only their metadata.

The Controller is also responsible for detecting storage node failures and ensuring the system *replication level* is maintained. In your DFS, every chunk will be replicated twice for a total of 3 duplicate chunks. This means if a system goes down, you can re-route retrievals to a backup copy. You'll also maintain the replication level by creating more copies in the event of a failure.

## Storage Node

Storage nodes are responsible for storing and retrieving file chunks. When a chunk is stored, it will be checksummed so on-disk corruption can be detected.

Some messages that your storage node could accept (although you are certainly free to design your own):

- Store chunk [File name, Chunk Number, Chunk Data]
- Retrieve chunk [File name, Chunk Number]

One alternative is creating unique identifiers for each chunk, in which case you wouldn't need the file name + chunk number combo. In that case, you'll have to think about how you inform clients on how to reconstruct the file correctly.

Finally, the storage nodes will send a **heartbeat** to the controller periodically. The heartbeat includes chunk metadata to keep the Controller up to date, while also letting it know that the node is still alive. Heartbeats should be sent every 5 seconds and only include the latest changes at the node, not an entire list of its files. However, the Controller can also ask the

storage nodes to send a complete file list (useful if the Controller failed and wants to rebuild its view of the system state). You can also include the amount of free space available at the node in your heartbeat messages so that the Controller has an idea of resource availability.

## Client

The client's main functions include:

- Breaking files into chunks, asking the Controller where to store them, and then sending them to the appropriate storage node(s). **Note**: Once the first chunk has been transferred to its destination storage node, that node will pass the chunk along in a pipeline fashion. The client should not send each chunk 3 times.
- Retrieving files in parallel. Each chunk in the file being retrieved will be requested and transferred on a separate thread. Once the chunks are retrieved, the file is reconstructed on the client machine.

The client will also be able to print out a list of files (retrieved from the Controller), and the total available disk space in the cluster (in GB).

## Tips and Resources

- Log early, log often! You can use a logging framework or just simple println() calls, but you should print copious log messages. For example, if a StorageNode goes down, the Controller should probably print a message acknowledging so. This will help you debug your system and it also makes grading interviews go smoothly.
- Here's an example of using Protocol Buffers for serialization and communication.
- Use the bass cluster (bass01 – bass24) to test your code in a distributed setting.
    - These nodes have the Protocol Buffers library installed as well as the `protoc` compiler.
    - To store your chunk data, use `/home2/$(whoami)`, where `$(whoami)` is your user name.

## Project Deliverables

This project will be worth 20% of your course grade (20 points). The deliverables include:

- **[2 pts]**: A brief design document (1 page). You may use UML diagrams, Vizio, OmniGraffle, or even pen and paper. This outlines:
    - Components of your DFS (this includes the components outlined above but might include other items that you think you'll need)

    - Design decisions (how big the chunks should be, how you will place replicas, etc…)
    - Messages the components will use to communicate
- **[6 pts]**: Control node implementation:
    - **[1]** File list and chunk list
    - **[1]** Storage node list
    - **[2]** Detecting a node failure
    - **[2]** Replica maintenance
- **[5 pts]**: Storage node implementation:
    - **[2]** Storing chunks and checksums on the local disks
    - **[2]** Detecting (and recovering from) file corruption
    - **[1]** Heartbeat messages
- **[5 pts]**: Client implementation:
    - **[2]** Storing files (chunk creation, determining appropriate servers)
    - **[2]** Retrieving files in parallel
    - **[1]** Viewing the file list and available disk space
- **[2 pts]**: Project retrospective document

Note: your system must be able to support **at least 10** active storage nodes. During grading, you will launch these components on the bass cluster.

## Milestones

Here's some milestones to guide your implementation:

- Week 1: design document complete, basic Protocol Buffer messaging implementation
- Week 2: client chunking functionality, controller data structures
- Week 3: storage node implementation, heartbeats, single-threaded client retrievals
- Week 4: failure detection and recovery, parallel retrievals
- Week 5: wrap-up, client file list/disk functions, retrospective

You are required to work alone on this project. However, you are certainly free to discuss the

project with your peers. We will also conduct in-class lab sessions where you can:

- Get help, discuss issues, think about your design
- Demonstrate working functionality early to receive partial credit for completed milestones. If you didn't understand a requirement or have an error in your logic, you will know early.

## Grading

You will have a one-on-one interview and code review to grade your assignment. You will demonstrate the required functionality and walk through your design.

I will deduct points if you violate any of the requirements listed in this document — for example, using an unauthorized external library. I may also deduct points for poor design and/or formatting; please use good development practices, break your code into separate classes based on functionality, and include comments in your source where appropriate.

## Changelog

- *10/17*: Added link to project retrospective document.
- *10/4*: Added user port assignment list.
- *9/7*: Added a small hint about storing files on the bass cluster.
- *9/6*: Added another bullet to the design document grade breakdown and added the due date: October 6.
- *9/6*: Added note about pipelining storage requests; client should only send each chunk once.
- *8/31*: Added tips and resources section
- *8/30*: Version 1.0 posted