



# UNIVERSITY of NEW ORLEANS

DEPARTMENT OF COMPUTER SCIENCE

---

## CSCI 6450: Principles of Distributed Systems

Instructor: *Vassil Roussev*

### Programming Project

#### The UNO Parallel File System (*punoFS*)

Assigned: Sep 22, 2009

#### **Due Dates**

- **Design & Implementation Plan: Thu, Oct 1**
- **Checkpoint #1: Thu, Oct 22**
- **Checkpoint #2: Thu, Nov 12**
- **Final Report & Demo: Thu, Dec 3**

#### **Goal**

The goal of the programming project is to develop a simplified version of a distributed file system which supports parallel I/O operations. The main objectives are to demonstrate scalability and fault-tolerance.

Your system will consist of four components as follows:

- *puno* Object server—maintains a flat object store;
- *puno* Metadata server—maintains the mapping of files to objects;
- *punoClient*—implements a POSIX-like file system API;
- *punoShell*—a simple command-line interface.

*punoShell* supports a very simple ftp-like interface with following commands, requirements, and assumptions:

- `put <file>` upload the given from the local file system to the *punoFS*;
- `get <file>` download the given from the *punoFS* to the local FS;
- `rm <file>` remove the given file from the *punoFS* (no questions asked!);
- `ls` list the contents of the *punoFS*, output should look like "`ls -l`";
- `df` report available *punoFS* disk space;
- `format` wipe all existing files and create an empty *punoFS* volume;

- `!<command>` execute `command` on the local system; that is `!ls -l` would yield the local listing on a Unix host;
- All commands must be read from the standard input; e.g., it should be possible to run a script in the following manner: `"punoShell < script.psh > output.txt"`;
- Every command must report its elapsed time in milliseconds on a new line in the format `"time: XXXXXms"`;
- The shell may be integrated in the same module as the client;
- At startup, the shell may take necessary input parameters of your choosing either from the command line or from an initialization file; e.g., `"punoShell param1 param2 < script.psh > output.txt"`.

## **Deliverables**

### ➤ ***Design & Implementation Plan: Thu, Oct 1***

The main tasks are to find a partner (optional) and to formulate a basic design & implementation plan. In particular you must decide on an implementation platform and schedule of work. In your plan you must specify realistic intermediate milestones that you plan to meet for checkpoints #1 and #2. These must include draft versions of your final report.

### ➤ **Checkpoint #1: Thu, Oct 22**

Intermediate report and demo.

### ➤ **Checkpoint #2: Thu, Nov 12**

Intermediate report and demo.

### ➤ **Final Report & Demo: Thu, Dec 3**

Final report and demo.

## **Requirements**

### *Your implementation must:*

- Implement an object-based model, a la GoogleFS/PanFS/pNFS;
- Be capable of distributing the storage over at least 4 chunk/object servers;
- Support volumes of at least 8GB;
- Support multiple readers/writers;
- Provides a POSIX-like file system API;
- Support a flat directory structure (sub-directories are strictly optional);

### *Your implementation should:*

- Scale up as more clients are added (aggregate throughput should increase);
- Have well-defined sharing semantics (specified in your design document);
- Be able to survive the failure of an object server;
- Completely heal (restore to pre-failure service level) within a reasonable amount of time as a failed object server comes back online;

*Your implementation will get extra credit if:*

- It can survive and recover from a metadata server crash (20%);
- Is among the top 3 in performance benchmark testing (15%,10%,5%, respectively);
- It can heal from a object server crash *while* continuing to serve clients (15%);
- Supports nested directories (10%);
- Come up with the best name for the system (5%).

### **Assumptions**

*You may assume that:*

- Only file name and size are mandatory file attributes to maintain;
- File name does not exceed 248 bytes;
- Security/access control is not a concern;
- The main concern is large files—small ones (KB) should be possible but the implementation need not be optimized for them;
- Optimizing for throughput is more important than optimizing latency;
- Object servers are always shutdown properly;
- Servers have enough time to initialize before any requests.