CoreGRID TR- Preliminary report

Author: Ada Lho-la Casanovas

Subject of research: Server Performance Analysis and Monitoring

Revised Objectives of the REP:

- Building a benchmark tool for server performance and monitoring.
- Designing an architecture of Session Storage Management patch for crash recovery in a Tomcat server.
- Performing experiments to test the behavior and recoverability of a Tomcat server under different load circumstances and injecting memory leaks.

I. iMLt: The Benchmark Tool

a. Briefing

We have designed and implemented a 100% Java distributed tool which is able to perform a stress test of a service under real load. In more detail, taking as input the traces of a real load or some concise description of a synthetic load, our tester generates a series of requests and issues these requests via a pool of distributed, emulated clients. Our design allows for a flexible configuration of the type of test as well as the type of service. Additionally it can be extended and tuned to fit the specific purposes of the user.

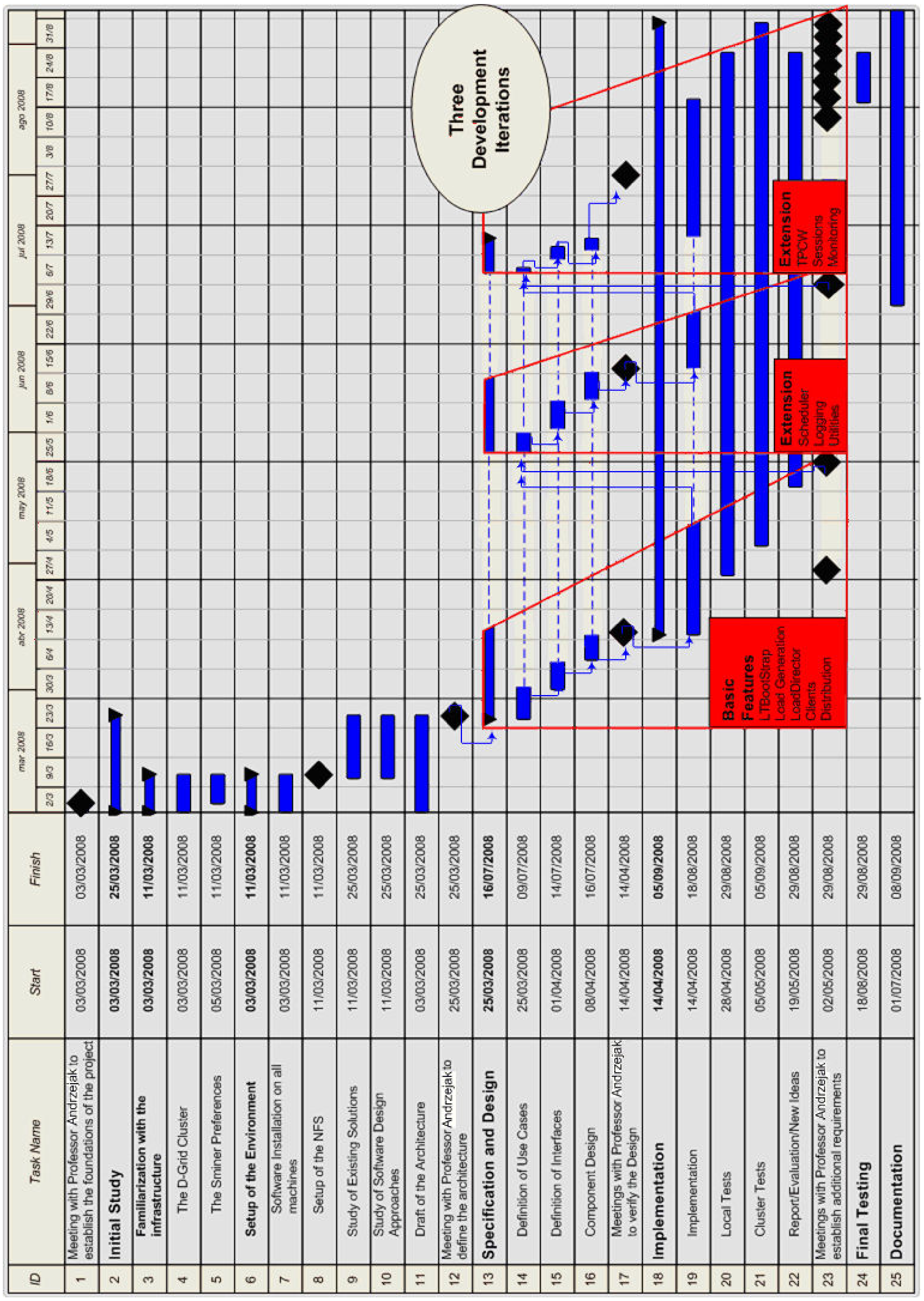
b. Steps of the project

The project has followed the standard software design steps, accompanied by several meetings with professor Andrzejak and Javier Alonso to specify and verify the requirements.

Steps:

- Study of the requirements
- Study of other stress and benchmark applications
- Preliminary Design
- Architecture and final Design
- Implementation
- Testing

Due to additional requirements the last 3 steps have been divided in 3 iterations. The resulting Gantt Diagram is the following figure



ID	Task Name	Start	Finish
1	Meeting with Professor Andrzejak to establish the foundations of the project	03/03/2008	03/03/2008
2	**Initial Study**	**03/03/2008**	**25/03/2008**
3	**Familiarization with the infrastructure**	**03/03/2008**	**11/03/2008**
4	The D-Grid Cluster	03/03/2008	11/03/2008
5	The Sminer Preferences	05/03/2008	11/03/2008
6	**Setup of the Environment**	**03/03/2008**	**11/03/2008**
7	Software Installation on all machines	03/03/2008	11/03/2008
8	Setup of the NFS	11/03/2008	11/03/2008
9	Study of Existing Solutions	11/03/2008	25/03/2008
10	Study of Software Design Approaches	11/03/2008	25/03/2008
11	Draft of the Architecture	03/03/2008	25/03/2008
12	Meeting with Professor Andrzejak to define the architecture	25/03/2008	25/03/2008
13	**Specification and Design**	**25/03/2008**	**16/07/2008**
14	Definition of Use Cases	25/03/2008	09/07/2008
15	Definition of Interfaces	01/04/2008	14/07/2008
16	Component Design	08/04/2008	16/07/2008
17	Meetings with Professor Andrzejak to verify the Design	14/04/2008	14/04/2008
18	**Implementation**	**14/04/2008**	**05/09/2008**
19	Implementation	14/04/2008	18/08/2008
20	Local Tests	28/04/2008	29/08/2008
21	Cluster Tests	05/05/2008	05/09/2008
22	Report/Evaluation/New Ideas	19/05/2008	29/08/2008
23	Meetings with Professor Andrzejak to establish additional requirements	02/05/2008	29/08/2008
24	**Final Testing**	**18/08/2008**	**29/08/2008**
25	**Documentation**	**01/07/2008**	**08/09/2008**

c. Architecture of iMLt.

Our stress tool has been built to accept any service client, that is, users can build a small Java program that executes the request for the service they wish to test, and anchor it to ours. With the schedule extracted from the input data we can generate requests using the anchored service client, emulating a real scenario. These request schedule will be distributed among different client machines to avoid performance issues.

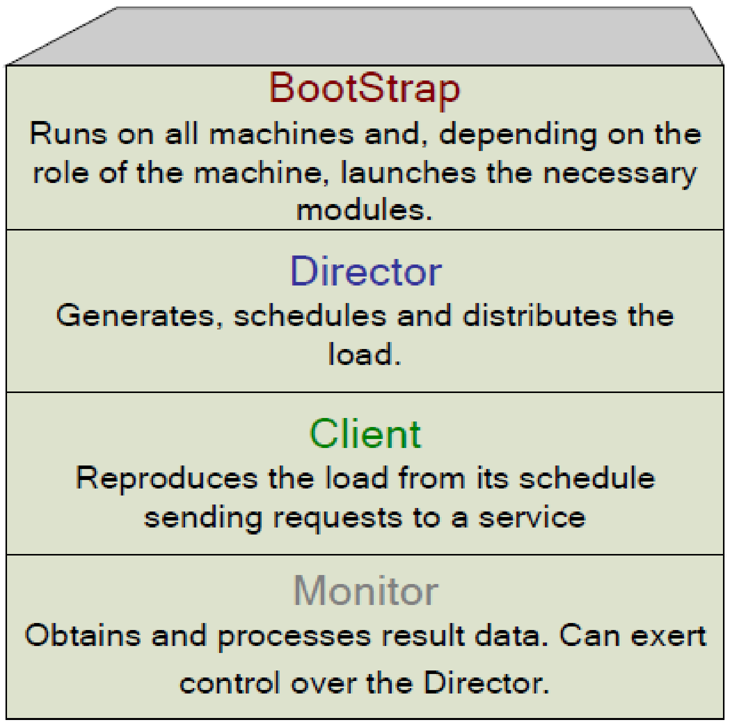

BootStrap
Runs on all machines and, depending on the role of the machine, launches the necessary modules.
Director
Generates, schedules and distributes the load.
Client
Reproduces the load from its schedule sending requests to a service
Monitor
Obtains and processes result data. Can exert control over the Director.

Figure 1: Roles in the iMLt Architecture.

The iMLt's architecture comprises four roles: BootStrap, Director, Client and Monitor. Figure 1 outlines their responsibilities in chronological order : the Tester will first BootStrap in all JVMs and determine their role, thus proceeding with the necessary steps to launch them. As a choice of design, both the Monitor and the Director run in the same JVM, several Client JVMs can be distributed locally or remotely. In the following paragraphs we will briefly define the basic features of each role as well as the interactions between them.

A more detailed view of the architecture is presented in Figure 2. Users configure the scenarios using an XML file and, if necessary, add other custom input (data on the load distribution, new implementations of iMLt key components, their webservice client). We offer interfaces to all key components: he LoadDirector will control the execution, the LoadGeneratior will create a schedule of load based on the user’s preferences, the Scheduler will distribute it, the Client will execute it and the Monitor will generate metrics extracted from logs. Communication and control is done via RMI services.

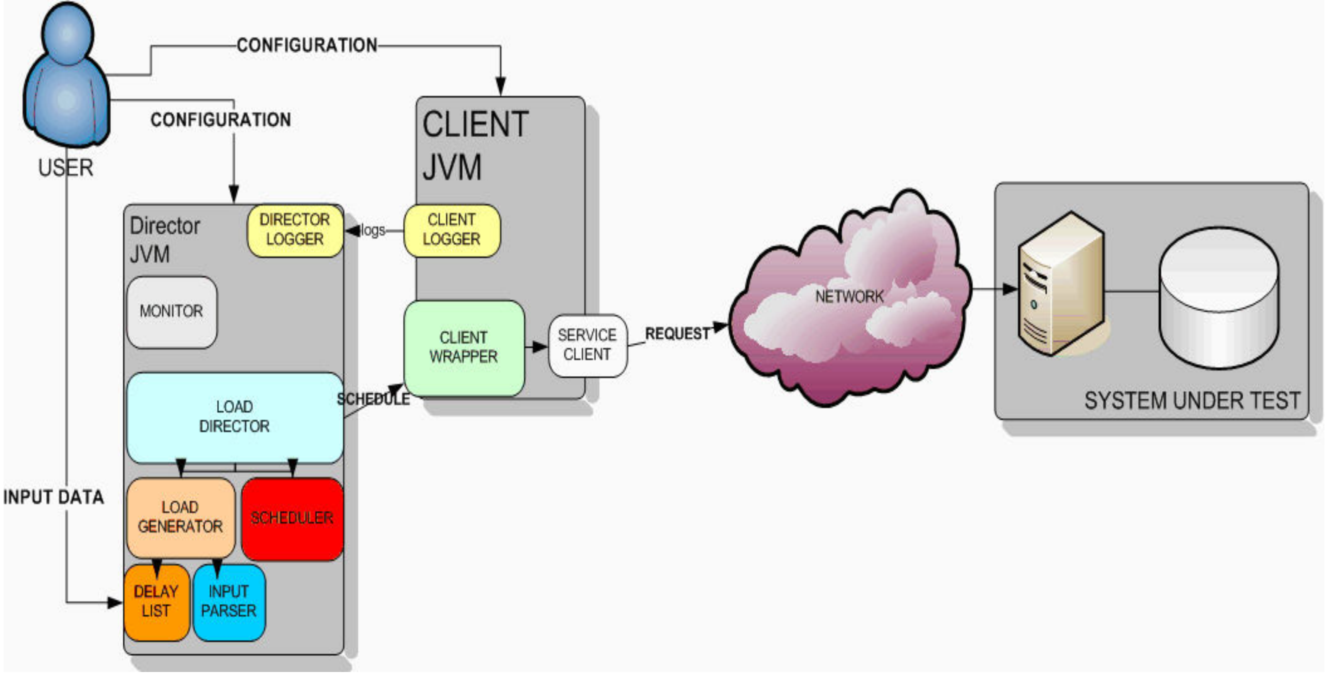


Figure 2: iMLt's Architecture

d. The TPC-W plugin.

We have used a Transaction Process Council Web (TPC-W) Benchmark java implementation(http://www.tpc.org/tpcw/default.asp), to design a plugin for this benchmark that enables us to use it with the iMLt. Therefore being able to emulate real requests to an e-book store using the TPC-W Emulated Browsers.

e. Progress of the Technical report

Implementation and Documentation of the iMLt architecture, related work and functionment is completed. In section III of this document we present the definition and schedule for the remaining Experiments using the iMLt to evaluate server aging.

II. Architecture of a Session Storage Management patch

We have designed a preliminary architecture for a possible Session Storage Management patch in a Tomcat server. This new SSM solution would consist in a 2 tier architecture using an external database to keep track of the activities of all sessions being served by a given server. Upon server crash, a recovery procedure would allow the same server or a clone to request this information from the database and continue serving the clients thus making the crash seemless.

For all Tomcat Contexts (applications) the session storage management architecture shown in Figure 3 will implement the following functions:

- Each time a session is created, accessed or modified, the SessionStorageManager will call functions of StoredSession .
- The StoredSession instance will create a SessionMessage and will send it to it's manager.
- The SessionStorageManager will add the message the SessionQueue
- The SessionQueue will wrap the message in a SessionEntryMsg .

- The SessionMsgSender will retrieve bulks of 5-10 messages from the queue in ByteArray format and send them
- The Storage Server can send ACK messages with several ID, captured by the SessionStorageReceiver, and forwarded to the appropriate SessionMsgReceiver.
- Unless it receives an ACK, a message will remain in the queue and be resent until the session expires.

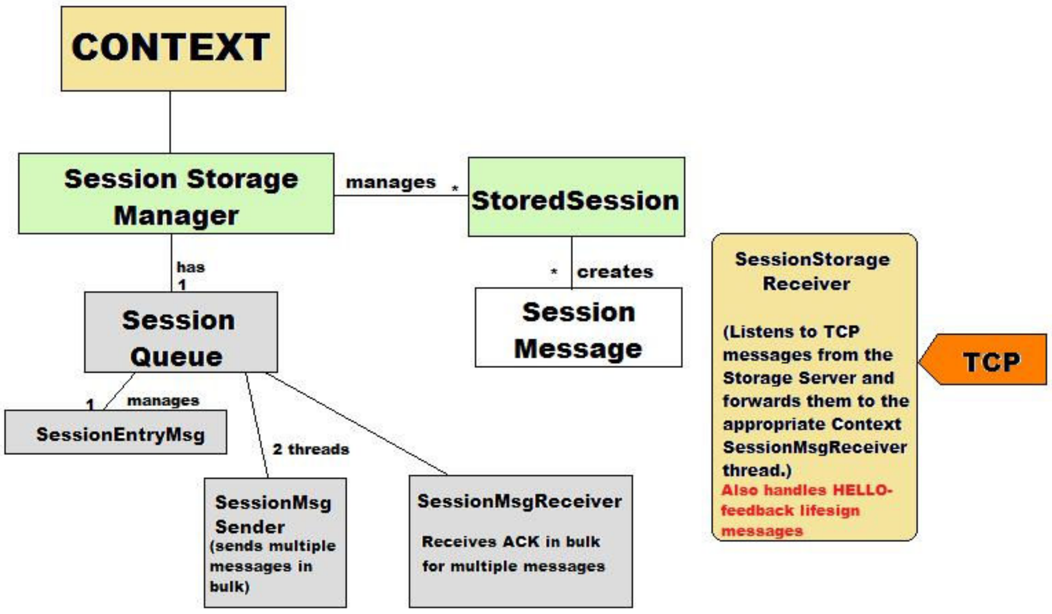

CONTEXT
Session Storage Manager
manages
*
StoredSession
has
1
Session Queue
*
creates
Session Message
1
manages
SessionEntryMsg
2 threads
SessionMsg Sender (sends multiple messages in bulk)
SessionMsgReceiver
Receives ACK in bulk for multiple messages
SessionStorage Receiver
(Listens to TCP messages from the Storage Server and forwards them to the appropriate Context SessionMsgReceiver thread.)
Also handles HELLO-feedback lifesign messages
TCP

Figure 3: SSM Architecture

The progress of the SSM project has been postponed outside the scope of the REP due to the global decision of giving priority to the iMLt implementation and usage.

III. Experiments using the iMLt

We are currently performing modifications of the iMLt functionalities in collaboration with Professor Andrzejak to allow for specific experiments on server aging by varying load, and dynamically injecting load for probing following the methods described in Andrzejak's NOMS08 paper *Using Machine Learning for non-intrusive Modeling and Prediction of Software Aging*.

Current Objectives:

- Modifying the iMLt metric extraction utilities to satisfy the experiment needs.
- Experiments injecting Memory Leaks.
- Experiments using probes.

Estimated Time to complete experiments 8 weeks.
Estimated time to complete the Technical Report: +1 weeks. Total : 9 weeks