CLIF is a Load Injection Framework

Bruno Dillenseger
France Télécom R&D
BP 98
38243 Meylan cedex, France
bruno.dillenseger@francetelecom.com

Emmanuel Cecchet
INRIA
655, avenue de l'Europe
38330 Montbonnot, France
emmanuel.cecchet@inrialpes.fr

ABSTRACT
This paper presents the architecture and preliminary development of Clif, a new project initiated by the ObjectWeb consortium in the field of benchmarking. Clif intends to provide a generic platform suitable for load injection and performance measurement on any system/protocol, using arbitrary load scenarios/profiles. As of current status, Clif consists of a distributed Java framework based on ObjectWeb's Fractal component model.

Categories and Subject Descriptors
C.4 [**Performance of Systems**]: *design studies, performance attributes, measurement techniques.*

General Terms
Load, Performance, Measurement, Design, Experimentation.

Keywords
Load injection, component, middleware, benchmarking.

1. ObjectWeb's benchmarking activities
ObjectWeb [7] is an international consortium fostering the development of open-source middleware for cutting-edge applications, such as Enterprise Application Integration, e-business, clustering, grid computing, managed services, etc. The evident necessity for a benchmarking action, and the wide use of Java resulted in the creation of the Java Middleware Open Benchmarking (JMOB) [4] meta-project.

One of the initial objective of JMOB was to make available open-source middleware benchmarks such as RUBiS [2] and provide support to the community to help users reproduce the results and improve the benchmarks. JMOB also became a natural place to share benchmarks and results on ObjectWeb middleware, and in particular on application servers (in a first step, the JOnAS J2EE application server [5]).

JMOB benchmarks are currently used to investigate impact of hardware configurations on performances, and in particular of new architectures (e.g. IA64). We are also working on the evaluation of distributed middleware executing on cluster architectures. When a single application server instance may require up to 5 client machines to reach its peak point, benchmarking a whole application server cluster becomes a real challenge from the load injection point of view. In fact, the load injection application may become more complex than the system to test.

In such a context, having ad-hoc client emulators for each and every benchmark is no more viable to evaluate large scale distributed systems and middleware. A real need for a generic distributed load injector has arisen from our benchmarking experience. In the very beginning of year 2003, France Telecom R&D and INRIA initiated the open source development of CLIF, a Java framework aimed at generating arbitrary load/traffic on arbitrary systems, via arbitrary invocation protocols.

2. CLIF: yet another benchmarking tool?
CLIF project started from an overview of existing projects. Many interesting projects and platforms were found but the conclusion was that, on the one hand, good commercial products were outstandingly both efficient and expensive, and on the other hand, that no open source project was really fitting both CLIF targets and ObjectWeb's approach in terms of software architecture.

CLIF functional requirements are the following:

- generic framework featuring hot swappable components and transparent deployment in a clean, sound architecture;
- distributed load generation for scalability;
- distributed test management with inexpensive runtime monitoring capabilities;
- optional user interface (in other words, support for both batch and interactive tests);
- efficient, powerful analysis tools for postmortem exploitation of test results, or even for runtime, automatic feedback on test parameters.

3. CLIF architecture

3.1 Fractal component model
Fractal [1] is a modular and extensible component model that can be used with various programming languages to design, implement, deploy and reconfigure various systems and applications, from operating systems to middleware platforms and to graphical user interfaces.

The goal of Fractal is to reduce the development, deployment and maintenance costs of software systems in general, and of ObjectWeb projects in particular. The Fractal model already uses some well known design patterns, such as separation of interface and implementation and, more generally, separation of concerns, in order to achieve this goal. There is also an ongoing research work to get even closer to this goal.

As far as CLIF is concerned, Fractal provides a relevant support for:

- designing, thanks to its uniformity (generalized single component structure and management for every service, whatever functional or technical), its recursion (components may be compositions of sub-components), and finally its general-purpose and minimalist approach;
- flexible runtime deployment and management thanks to a reflection capability on components, compositions and bindings.

3.2 CLIF, a Fractal-based distributed architecture

CLIF has been designed using the Fractal model, and implemented using Julia, a Java implementation of Fractal, and Fractal-RMI, an RMI-based extension for distribution support.

CLIF's distributed architecture is composed of a single supervision **console** component, for test management (deployment, control, monitoring), and an arbitrary number of "**CLIF server**" components (see Figure 1). Those servers are empty, minimal Fractal components, in which the console may create a **load injector** component and/or a **resource probe** component. The former component is in charge of hosting and executing test scenarios, while the latter measures the consumption of system resources (related to CPU, memory, swap, network, disk, etc.). CLIF servers register themselves in a Fractal-RMI registry to be found and reached by the console.

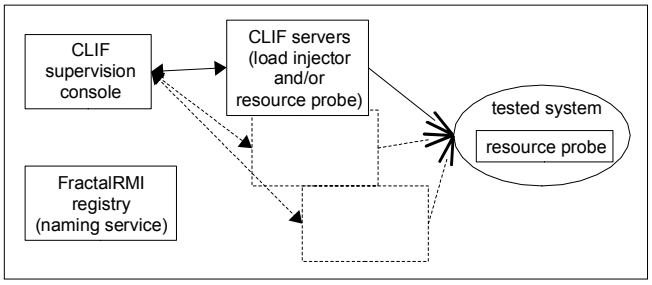

Figure 1. CLIF architecture: a distributed system of Fractal components

Resource probes shall be deployed:

- on nodes hosting the tested system, typically to see if observed throughput saturation or error occurrence is due to some resource shortage, and then to improve the system (hardware or software layers);
- as well as on the load injectors, to detect a saturation of the injection system, which is a typical problem of load injection tools in most real situations.

4. CLIF implementation

4.1 Main components

Besides console and server components, other main Clif components are:

- a **storage** component, which is responsible for storing test data;
- a **collector** component responsible for transporting test data to the storage component;
- an **analyzer** component supporting exploitation of test data;
- a number of **scenario** components, one per load injector, responsible for actually invoking the tested system according to their own specific load injection specification, as well as delivering test data to the collector component (response times, error occurrences, results, etc.).

4.2 Focus on scenario component

The scenario components are deployed and controlled by the console among every Clif server acting as load injector. As a consequence, a full test is the combination of all deployed test scenarios. Each scenario may be arbitrarily complex or simple. The simplest scenarios may consist in repeatedly invoking a couple of operations on a tested server with the definition of load level ramps or pikes. More complex scenarios may simulate groups of users with one or several behavior types, and even include server-local or injection system-wide synchronization.

As a framework, Clif is independent from scenarios implementation and complexity. How they can be defined and executed is out of Clif's scope, but the services they offer or require as Fractal components are clearly specified. On the one hand, a scenario component offers a management service (specified by a Java interface) to control its initialization and execution (start, stop, suspend, resume and join, i.e. wait for the scenario execution end). On the other hand, the scenario component makes use of two services: one to give test data to the collector component, and a second one to inform about the scenario status (initializing, ready, executing, etc.).

4.3 Test data

Clif collects and stores several kinds of test data:

- events about the life cycle of each scenario and the overall test (deployed, started, suspended, aborted, etc.);
- alarms with severity levels;
- performance measurements, including the action name, the injector name, the response time, a success flag, the iteration number (when applicable), the session identifier

uniquely designating a virtual user (when applicable), a comment string and a result object;
- resource static information describing every node running a resource probe: number and type of CPU, memory size, operating system, network adapter, etc.;
- resource consumption information for every node running a resource probe: CPU usage, context switches, page swapping rate, network errors, network packet rate, network packet collisions, disk transfer rate, etc.

All these data records are locally time-stamped by each node. The clocks of these nodes are synchronized using the Network Time Protocol. Implementing distributed causality between all these data records would be a possible improvement, but a great care should be taken not to overload the system. We believe that NTP precision is sufficient for the kind of data exploitation we are currently thinking about. A deep correlation analysis would, of course, require to take into account the possible slight shift between clocks.

5. Clif project status

A first version of the framework is ready (Clif server, console), together with a minimal graphical user interface and a few utilities, such as an HTML parser or a state transition matrix with think time directly inspired from the RUBiS benchmark. The resource probe has been implemented for Linux only, by making use of `/proc` information.

No true collection and storage components are ready yet, but a simple local file based system is available. We are working on a Message-Oriented Middleware and J2EE based architecture to asynchronously collect data (to avoid disturbing and overloading the testing system), and to store these data in a database, which is probably the best way to exploit great benchmarking campaigns.

The alarms are not available yet. A preliminary work has been done in the form of a distributed log system for Fractal based on the Monolog [6] logging API, but it is still under development. Then, an integration to Clif console will be made, so that a response can be automatically made by Clif when alarms of a given severity or user-defined alarms occur (e.g. display the alarm message, suspend, abort or restart the test, etc.)

Future work deals with:

- analysis tools to actually perform and help data exploitation from the database (report generation, data-mining techniques, event correlation, feedback on tests definitions and parameters...);
- scenario components for a variety of load profiles (ramps, peaks, coarse-grain overall load or fine-grain user simulation or replay of recorded real user sessions...);
- user input tools (graphical, scripts...);
- support libraries for typical targets (HTTP/HTTPS, LDAP, JDBC, FTP, XML-RPC, IP sockets...);
- overall management of full benchmarking campaigns.

6. Conclusion

Clif is a Load Injection Framework, implemented in Java following the Fractal component model. It is a starting open source initiative of the ObjectWeb community in the field of benchmarking. The specific aim of this project is to define a smart infrastructure open to any target system, any user skill, and any load profile including high loads. Being a new project, a preliminary version of the load injection architecture is ready, but a number of components are currently lacking to make it actually usable for test campaigns. Some of them are under development, others are going to be implemented or designed.

7. Acknowledgment

Thanks go to Julien Buret and Nicolas Droze for their work on Clif. Thanks go also to other comparable projects, especially RUBiS and Apache Jmeter [3] from which we got inspiration and experience – and possibly someday pieces of code? :-)

Visit Clif at http://clif.forge.objectweb.org/.

8. REFERENCES

[1] E. Bruneton, T. Coupaye, J.-B. Stefani – Recursive and Dynamic Software Composition with Sharing. Seventh Int. Workshop on Component-Oriented Programming (WCOP02), ECOOP 2002, Malaga, Spain.

[2] Emmanuel Cecchet, Julie Marguerite and Willy Zwaenepoel – Performance and scalability of EJB applications – Proceedings of OOPSLA'02, November 2002.

[3] Jmeter – http://jakarta.apache.org/jmeter/

[4] JMOB – Java Middleware Open Benchmarking – http://jmob.objectweb.org.

[5] JOnAS: Java Open Application Server – http://www.objectweb.org/jonas.

[6] Monolog – http://monolog.objectweb.org.

[7] ObjectWeb consortium. – http://www.objectweb.org.