Benchmarking Environment (BEEN)

Project supervisor: Tomáš Kalibera, KSI, MFF UK (tomas.kalibera@mff.cuni.cz)
Number of students: 4-6

Overview

The aim of the project will be to create a highly configurable and modular environment for benchmarking of applications, with special focus on middleware benchmarks.

The core functionality of the environment will include automatic middleware acquisition, compilation, benchmark acquisition and compilation, allocation of computers, deployment, execution, monitoring, collecting results, results conversion and storage in results repository.

The environment will include a tool for generating graphical reports of results and a tool for automatic analysis of results. Implementation of these tools within the scope of a students project is optional.

The environment will support benchmarking suites like Xampler [1], developed by Distributed Systems Research Group (http://nenya.ms.mff.cuni.cz).

Requirements

The environment must be very flexible to support a wide range of middleware benchmarks, ranging from CORBA micro-benchmarks to complex model application benchmarks using EJB. The environment must be easy to install and administer on different types of hardware and operating systems with none or minimal requirements on additional software. Supported target platforms must include Linux, Solaris and Microsoft Windows. Programming language will be Java, with an exception of interfacing with operating system not available through Java environment. Parts of optional regression detector module may be written in a language for statistical computing, preferably R Language.

The environment must not interfere with the running benchmarks to avoid possible distortion of benchmark results.

All participants of the project should have good programming skills and experience with the Java language. The development language is English, for all documents, documentation and source code. Intensive work towards project completion within minimum time and tight cooperation with the other benchmarking projects of the Distributed Systems Research Group is expected.

Architecture

Benchmark execution environment.

Core system. The core of the system will take care of locating, deployment, life-cycle, configuration, monitoring and interoperability of modules of the environment. All configuration will be XML based, modules will be Java classes. As the system will need to run modules on multiple computers, the core system will also include support for distribution, preferably using Java RMI.

Core modules. The core modules will provide general tasks for functional modules of the system, including logging, retrieving resources via ftp, scp, http and other protocols, interface to CVS, accessing files in archives, remote execution of commands, etc. Third party libraries may be used for some of the tasks.

Allocator module. The allocator will be responsible for allocating hosts to benchmarks based on requirements stored in benchmark configuration and on system information stored in hosts configuration and information fetched automatically from the hosts. At the same the allocator module will allocate hosts for compilation of benchmarks and middleware, depending on available compilers and hardware platforms.

Builder module. The builder module will control building of benchmarks and middleware, it will use the allocator module to select hosts on which the build can be accomplished, transfer all required source files, run and monitor the build process, and collect results.

Benchmark execution module. This module will deploy and run benchmarks on hosts selected by allocator module, monitor their execution and collect the results. The monitoring must be accomplished with minimal interference on benchmark results.

Results repository.

The results repository will store results of all benchmarks. The benchmarks results will have a proprietary and common format. The repository will provide functions for intelligent retrieval of results stored in the common format, including filtering based on observation values, benchmark configuration, characteristics of the system under test, as well as statistical functions and other transformations. As the benchmarks produce gigabytes of results, the repository will have to use a feasible compression mechanism.

Report generator (Optional).

The report generator will produce graphical and textual reports of benchmark results stored in results repository. It will support plugins for generation of proprietary reports based on proprietary benchmark data as well as it will use the common format of results to generate reports comparing results based on user defined filters (such as compiler type, CPU frequency or relative variation of observations).

Regression detector (Optional).

The regression detector will try to automatically detect performance regressions in middleware by comparing benchmark results of consecutive versions of middleware, as described in [2]. Parts of the detector may be written in a language for statistical computations, such as R Language.

References

[1] Xampler, http://nenya.ms.mff.cuni.cz/benchmark
[2] Bulej L., Kalibera T., Tůma P.: Regression Benchmarking with Simple Middleware Benchmarks, in proceedings of IPCCC 2004, IEEE Computer Society, pp. 771-776
[3] RUBiS, http://rubis.objectweb.org