2003

Towards Definitive Benchmarking of Algorithm Performance

Andrew Lim
Hong Kong University of Science and Technology, iealim@ust.hk

Wee-Chong Oon
National University of Singapore, oonwc@comp.nus.edu.sg

Wenbin Zhu
National University of Singapore, zhuwb@comp.nus.edu.sg

Towards Definitive Benchmarking of Algorithm Performance

Andrew Lim[1], Wee-Chong Oon[2], Wenbin Zhu[2]

[1]Department of IEEM, Hong Kong University of Sci and Technology
Clear Water Bay, Kowloon, Hong Kong
Tel: +852 23587116; Email: iealim@ust.hk; Fax: +852 23580062

[2]Department of Computer Science, National University of Singapore
3 Science Drive 2, Singapore 117543
Tel: +65 68746891; Email: {oonwc, zhuwb}@comp.nus.edu.sg; Fax: +65 67794580

Abstract

One of the primary methods employed by researchers to judge the merits of new heuristics and algorithms is to run them on accepted benchmark test cases and comparing their performance against the existing approaches. Such test cases can be either generated or pre-defined, and both approaches have their shortcomings. Generated data may be accidentally or deliberately skewed to favor the algorithm being tested, and the exact data is usually unavailable to other researchers; pre-defined benchmarks may become outdated. This paper describes a secure online benchmark facility called the ***Benchmark Server****, which would store and run submitted programs in different languages on standard benchmark test cases for different problems and generate the performance statistics. With carefully chosen and up-to-date test cases, the Benchmark Server could provide researchers with the definitive means to compare their new methods with the best existing methods using the latest data.*

Keywords: Benchmarking of algorithms, Web-based Benchmark Server

1 Introduction

Across computer science research, the evaluation of a new heuristic or algorithm is primarily dependent on its performance when compared to other existing approaches when tested on representative (i.e. benchmark) data. In general, benchmark data is either pre-defined or generated. In the current research environment, both have significant shortcomings as the definitive measure of an algorithm's performance.

Pre-defined benchmarks are specific test cases that are designed to be representative examples of a particular problem and made available to all. Since previous research on the problem will be based on these benchmarks, new researchers need not implement the old methods. Instead, they can evaluate these methods based on published results. However, such

benchmarks will eventually become outdated. For example, the standard benchmark for the Vehicle Routing Problem with Time Windows (VRPTW) has long been the 56 test cases of 100 customers devised by Solomon (Solomon 1987). Given today's increased computing power and technology, Solomon's test cases are no longer sufficient. A well-known extension has been proposed that includes instances with up to 1000 customers (Homberger 1999). Unfortunately, researchers that adopt this new problem set cannot compare their new methods with existing ones unless they implement the old methods and run them on the new test data.

For most problems, however, no pre-defined benchmarks exist. As a result, researchers have no choice but to generate their own representative test data. This is clumsy and inexact for several reasons. Firstly, the generation of such test cases can be difficult, time-consuming and challenging (Hall and Posner 2001), and the generated data may be erroneous or insufficient. Secondly, researchers seldom reveal the actual data used for their experiments as the data may be confidential or sensitive, or too large. Instead, they describe the data's characteristics in general terms. For instance, a typical test set for a Shortest Path Problem algorithm might be described as "20 undirected Euclidean random graphs of 100 vertices and 500 edges". This makes their claims difficult to verify or disprove, and indeed unscrupulous researchers may deliberately choose favorable test cases. Elevated performance claims are especially difficult to prove if the approach involves some random element (like Simulated Annealing) or training phase (like Neural Networks). Thirdly, the researcher must implement all the other approaches that he wishes to test his own approach against. As the researcher may not be familiar with the other approaches, this is another time-consuming and error-prone process.

There are other problems with the current benchmarking process. The actual running of the program on multiple test cases can be time-consuming and tedious. Researchers also often summarize algorithm performance using some overall measure like average performance or best *n* results, but may (inadvertently or deliberately) omit some critical measure like worst-case performance. In summary, the weaknesses of current benchmarking practice stem from (1) lack of carefully designed, consistent and current data; (2) incomplete or misleading performance evaluation statistics; and (3) implementation difficulties.

Benchmarking of algorithms is somewhat related to the automatic grading of programming assignments. A number of work has been done to automate this process (Forsythe and Wirth 1965, Urs von Matt 1994, Joy and Luck 1995, Jackson 1996, Leal and Moreira 1998, Kurnia, Cheang and Lim 2001, and Cheang, Kurnia, Lim and Oon 2003).

In the rest of the paper, we will present the unique challenges in automating the process of benchmarking algorithms. Section 2 presents the system architecture and design, and related technical issues, including fairness, scalability, security, fraud prevention and performance. Section 3 addresses our methodology and issues in deployment. Section 4 provides the conclusion.

2 The Benchmark Server

The Benchmark Server (working title) is a secure online distributed system that promises to overcome all the above weaknesses of current benchmarking practice (See Section 1 and Zhu 2002). Its core service is to securely run submitted programs on pre-defined benchmark data, and then return the appropriate performance statistics. The pre-defined benchmarks will be a combination of revealed data and statistically similar secret data. The performance of submitted programs will be ranked against all other submissions according to various criteria, allowing the researcher to easily judge his program's relative ability. All programs will be stored, and made available for public scrutiny upon permission of the submitter. This is an extension of the Online Judge system (Kurnia, Cheang and Lim 2001, and Cheang, Kurnia, Lim and Oon 2003) that was used as an automated programming assignment grader.

The Benchmark Server is composed of (1) a *main server* that controls the running of the programs; (2) multiple *computing servers* that perform program executions and benchmarking; (3) an *information storage* containing problem, benchmark, submission and server information; and (4) a *user interface* allowing user registration, program submission and statistics display. Each component is further divided into subcomponents for modularity and maintainability. Figure 1 shows the system architecture.

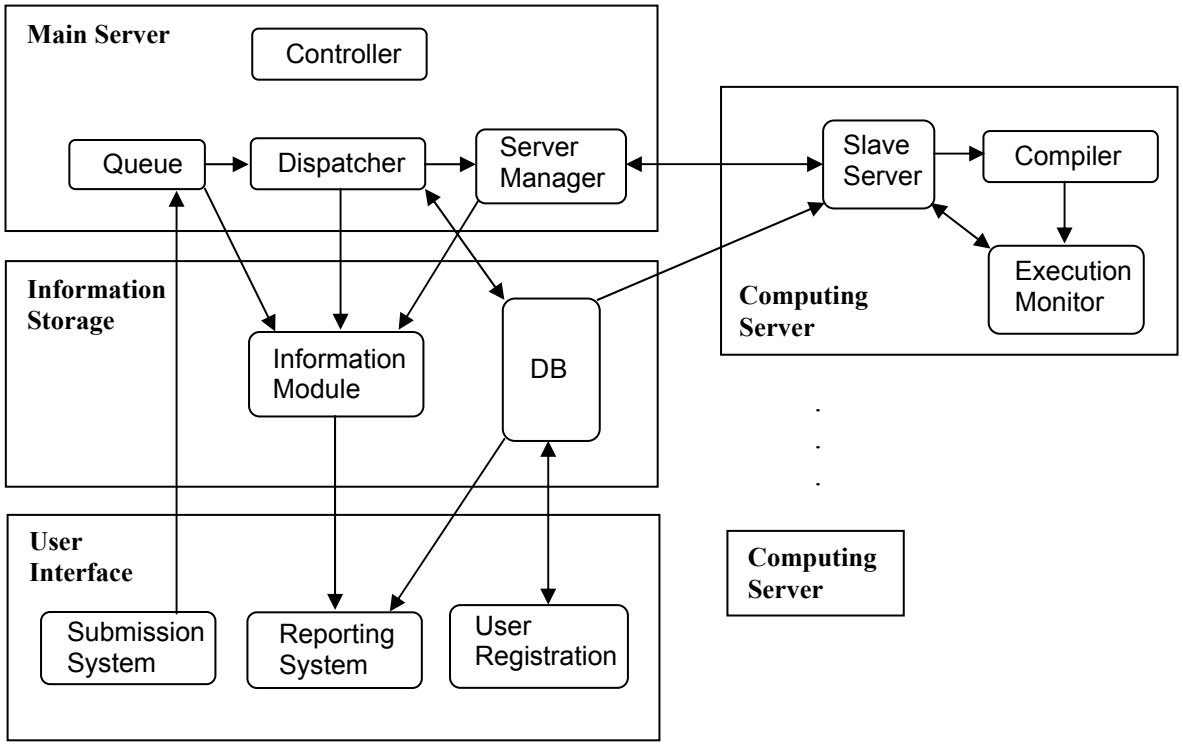

Figure 1: System Architecture

A new user registers onto the Benchmark Server via the *user registration* subcomponent. This information is stored in the *database*. The user can now submit programs through the *submission system*, which will be inserted into the priority *queue* in the main server. These programs are passed to the *dispatcher*, which uses a rule-based system to control the treatment of the target program (e.g. which benchmarks to load, which compiler to use). The appropriate benchmarks and problem information is retrieved from the database. The *server manager* then transmits the program to a free computing server if available. The actions of the main server are overseen by a *controller*, and real-time information like the current status of all the computing servers and the queue is stored in a separate *information module*. When a free *slave server* receives a submitted program, it is compiled using the appropriate *compiler*. The *execution monitor* records the performance statistics, which are saved onto the database. Finally, these statistics are displayed using the *reporting system*.

The aim of the Benchmark Server is to provide the research community with the definitive source of up-to-date benchmarking for a wide variety of problems, and the tools to perform such testing in a thorough, consistent and convenient manner. To achieve this aim, a number of basic requirements must be addressed. The rest of this section briefly describes how the Benchmark Server fulfils these requirements.

2.1 Security

Security is of paramount importance to any Internet venture, and is especially so for the Benchmark Server as it involves running external programs. Therefore, several measures have been taken to ensure the highest level of system security.

A malicious program can potentially crash the entire system. To prevent this, all submitted source code is scanned for restricted system or function calls, such as those that attempt to kill other processes or make network connections. Compiled programs are executed in a sandbox under restricted user permission allowing limited memory resources and CPU time. Furthermore, writes are only permitted to standard output. Finally, the main server and computing servers are situated on different machines. Therefore, a malicious program can only crash the particular computing server it is on, without affecting other submissions or benchmark data.

A submitted program may attempt to elevate performance values by connecting to a supercomputer, performing the tests on it, and then transmitting the results back to the program. To prevent this, all computing servers are on a private network with no Internet access. Firewalls are also installed on both the main server and computing servers. These measures also prevent submitted programs from transmitting secret benchmark test cases to external locations. Without knowing the secret test cases, a user cannot cheat by pre-processing the results offline.

Other security measures include a priority queue system that takes into account the number of submissions from each user (to prevent one user from flooding the queue with multiple submissions), and computing server registration requiring a 64-bit integer authorization token (to prevent external servers from impersonating a computing server and transmitting false

benchmark results). While the security of the system cannot be 100% ascertained until it is open to actual use, we believe that the Benchmark Server is immune to malicious submissions.

2.2 Computing Server Maintenance

The Benchmark Server can support any number of computing servers, and the workload is distributed evenly among all available servers. Should any particular server fail, it will simply be removed from the pool of available servers without affecting the rest of the system.

Computing servers are organized into server groups. All machines within a server group have identical hardware. The system administrator can designate problems to server groups so that all tests on a problem are done on machines with identical configurations. In this way, new machines can be added to the system as new server groups.

2.3 Multiple Language Support

The Benchmark Server defines a set of interfaces for programming language support. A new language is added by implementing an interface for it. For example, the Java language can be added by implementing

- A Compiler interface that will invoke an external compiler (e.g. `javac`) to compile a java program and store the resulting classes in a specified directory; and
- A CmdLine interface, which returns a string specifying how a compiled program is invoked. For Java, the string returned would be

```
java –classpath run_dir Main
```

2.4 Problem Management

Each problem is implemented as a separate package. The Benchmark Server provides tools to form packages containing the problem definition; its benchmark test cases; a verifier program that verifies solution correctness; and an evaluator program that evaluates the quality of a solution. These tools will be released to the public, allowing the proposal and definition of new problems.

3 Objectives and Research Methodology

It is hoped that the Benchmark Server will emerge as a useful tool for researchers, academic referees, programmers and students. Researchers will be able to use the Benchmark Server to find the current best approaches to various problems. By submitting their implementations, they can directly compare their work with existing algorithms on the exact same test cases and machines. Similarly, academic referees can verify researchers' claims via the Benchmark Server. Programmers of practical applications will be able to refer to the source code

of actual implementations of the best algorithms. Finally, the Benchmark Server can serve as a valuable learning tool for students.

However, this can only come to fruition if the Benchmark Server achieves widespread acceptance such that the authorities on the various problems submit their algorithms. While the Benchmark Server has met or will meet all the basic requirements, the natural resistance that exists against the adoption of new technology (Canton et al 1999) must be overcome. The objective of this research is to achieve this acceptance.

An important task is to choose appropriate problems and test sets for the initial release of the Benchmark Server. While it is simple to generate or devise test cases, it is difficult to justify that these test cases are sufficient and fair. Proper benchmark test cases is crucial for correct evaluation of algorithms, especially for approaches with random or training elements like Neural Networks (Flexer 1995, Prechelt 1996) For the initial implementation, we will likely choose well-known problems of wide interest with established benchmark test cases. The aforementioned VRPTW is one likely candidate. In the future, online communities can be set up for arriving at a consensus on the appropriate definitive benchmarks.

When the appropriate problems have been chosen, it would be useful to implement some of the common algorithms on them. For VRPTW, possible implementations include Genetic Algorithm, Simulated Annealing or some greedy heuristic. While these implementations may not be optimal, they can serve as useful baseline comparisons. Furthermore, since the source code for implementations will be available for scrutiny, users can submit improved versions to update the performance statistics.

The presentation of statistics must also be carefully considered. The performance statistics of submitted algorithms must be presented in a useful and convenient manner that allows easy comparison on various aspects of the algorithms. The most useful statistics will be problem-dependent, and must be chosen and presented with expert knowledge. Ideally, authorities on each problem should be consulted on the most important aspects to be measured and the most useful presentation formats. Furthermore, as the number of submissions increase, information overload must be prevented so that the statistics will not be cluttered by information on irrelevant algorithms.

Privacy is an important issue. There may be users who wish to test their approaches against the existing ones, but do not wish to divulge it. We plan to address this issue by allowing three types of submissions: source code, executables or results. Source code submission allows others to scrutinize the submitted code, and is the most useful of the three. If only executables in binary form are submitted, the Benchmark Server can still test them, but there is a danger of elevated performance due to pre-processing or other such techniques. Alternatively, if the user does not wish his program to be available in any form to the public but merely wishes to advertise its capability, he can simply submit results. These will be published on the web site, but with the disclaimer that the Benchmark Server has not verified it in any way.

Like any new venture, advertising will be crucial to the success of the Benchmark Server. The current plan is to provide it as a free service to academic institutions to encourage

widespread adoption. After the system is completed and thoroughly tested in-house, we hope to advertise it in leading journals and conferences across computer science. The main aim is to let all researchers know that there is a facility for up-to-date benchmarking available.

Many other value-added services are also possible. The Benchmark Server serves as the foundation to set up information repositories on the various problems. When a user checks up on the latest algorithms to a problem, web links can direct him to sites that contain the details of these approaches as well as the problem itself. Utilities can also be provided to create images of statistical information that can be downloaded and used in reports. The Benchmark Server can also serve as a protection of intellectual property. All submissions will be time-stamped and stored so that the submitter of the program can prove the originality of his idea. An online research community can be set up for the purpose of discussing problems, defining new benchmarks, proposing interesting problems and the general exchange of ideas.

We believe that the Benchmark Server has the potential to provide an invaluable service to the computer science community, and must be carefully managed to make this vision a reality.

4 Conclusion

The Benchmark Server makes use of the global connectivity of the Internet to overcome the pitfalls of current benchmarking practice. It allows researchers to submit the best implementations of their algorithms to a worldwide and up-to-date site to be tested on carefully devised definitive benchmark test cases, so that they can be compared to other algorithms when run on identical machines using identical resources. It will provide detailed and thorough statistics for easier comparison, and source code for perusal and adoption. With careful organization of information, the Benchmark Server can be a powerful educational tool for students and practical programmers.

We are currently in the process of choosing appropriate problems and benchmarks for implementation, and deciding the best methods of displaying performance statistics. In-house testing should be completed about 2 months from this time of writing, and we are also seeking to procure more machines for the system. We hope that the Benchmark Server will become the definitive source of benchmarking information on algorithm performance, and be enthusiastically adopted by the computer science community in the near future.

Acknowledgements

The paper is supported in part by the funding HIA02/03.EG04 (HKUST) and DAG02/03.EG07 (HKUST).

References

Canton, E. J. F.; de Groot, H. L. F; and Nahuis, R. "Vested Interests and Resistance to Technology Adoption", *CentER Discussion Paper*, no. 99106, 1999.

Cheang, B; Kurnia, A.; Lim, A. and Oon, W. "On Automated Grading of Programming Assignments in an Academic Institution", *Computers and Education*, accepted for publication 2003.

Flexer, A. "Statistical Evaluation of Neural Network Experiments: Minimum Requirements and Current Practice", Technical Report*, Proceedings of the Thirteenth European Meeting on Cybernetics and Systems Research*, R. Trappl (ed.), 1995.

Forsythe, G. E.; Wirth, N. "Automatic grading programs". *Communication of the ACM*, (8):275--278, 1965.

Hall, N.G.; Posner, M.E., "Generating experimental data for computational testing with machine scheduling applications", *Operations Research*, 49 (7), pp. 854-865, 2001.

Homberger, J. "Extended SOLOMON's VRPTW Instances", http://www.fernuni-hagen.de/WINF/touren/inhalte/probinst.htm

Jackson, D. "A software system for grading student computer programs", *Computers & Education*, v.27 n.3-4, p.171-180, Dec. 1996

Joy, M.S.; Luck, M. "On-line submission and testing of programming assignments". *Innovations in Computing Teaching,* pages 97-103, 1995.

Leal, J., P.; Moreira, N. "Automatic grading of programming exercises". *Technical Report DCC-98-4, DCC-FC&LIACC, UP,* June 1998

Kurnia, A.; Cheang, B and Lim, A "Online Judge", *Computers and Education*, 36, pp 299-315, 2001.

Prechelt, L. "A Quantitative Study of Experimental Evaluations of Neural Network Learning Algorithms: Current Research Practice", *Neural Networks* 9(3), pp. 457-462, April 1996

Solomon, M.M. " Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints", *Operations Research*, no. 35, pp. 254-265, 1987.

Urs von Matt, "Kassandra, the automatic grading system". *SIGCUE*, (22):26-40, 1994

Zhu, W. "The Benchmark Server", *Final Year Project report*, National University of Singapore, 2002.