Atlas: An Intelligent, Performant Framework for Web-Based Grid Computing

Sachith Gullapalli
Yale University
51 Prospect Street
New Haven, CT, USA 06511
sachith.gullapalli@yale.edu

ABSTRACT

This paper presents Atlas, a distributed computing system that allows for collaboration over Internet browsers. It allows users to donate the wasted processing power from their Internet-connected machines to help researchers and companies compute difficult tasks. The platform aims at maintaining similar speeds to available cloud computing services while running these tasks, while doing so at a lower cost. In order to do so, Atlas minimizes the amount of time needed per computation and intelligently distributes jobs by utilizing user browsing patterns. Benchmarks demonstrate that Atlas may be a viable alternative to existing platforms.

CCS Concepts

•Human-centered computing → Collaborative and social computing; •Computer systems organization → Grid computing; •Software and its engineering → Grid computing;

Keywords

grid computing, volunteer computing, OpenMP

1. INTRODUCTION

The world's most popular websites receive millions of unique visitors on any given day. Because most people have very powerful computers relative to the demands placed by website rendering, this represents an enormous amount of untapped computing power.

What if this untapped computing power could be used to solve research problems? In this paper, we discuss Atlas - a distributed computing system that allows for collaboration over Internet browsers. With this service, parallel research problems expressible in the OpenMP framework are executed across all connected nodes in a network, with each node handling a small piece of the total overall computation. Through the utilization of asm.js, we demonstrate greatly improved performance relative to previous work.

FSE'16, November 13–18, 2016, Seattle, WA, USA
ACM. 978-1-4503-4218-6/16/11...$15.00
http://dx.doi.org/10.1145/2950290.2983980

2. BACKGROUND

In a 2007 paper, Boldrin, Taddia, and Mazzini describe a solution for using web browsers as clients for distributed computing [1], utilizing Javascript and AJAX technologies to send subproblems one at a time from a server to a client, where computation takes place to resolve each subproblem. Finally, a single result is returned back to the server.

More recently, the Capataz framework was developed, which allows for web browsers on multiple devices to contribute to the execution of distributed algorithms [3] The system significantly improved upon the Boldrin method by providing a method for bundling different jobs together in order to minimize the amount of time spent computing solutions for similar subproblems. Analysis showed that the framework provided a viable method for distributed computing, and the job-bundling feature reduced execution time compared to previous systems.

Gray et al. propose a system utilizing a Weibull distribution to model dwell time on Web sites [4]. Because the probability that a browser session ends becomes smaller as the time after page load increases, they adaptively batch different jobs together in order to minimize the amount of time spent computing solutions for similar subproblems. This job-batching feature significantly reduced execution time compared to the naive approach used by previous systems.

The major bottleneck presented by these systems is speed of computation. Atlas significantly improves upon these systems by removing the requirement for input algorithms to be coded using Javascript. Even for simple programs, the physical time a block of Javascript code takes to run is significantly slower than an equivalent program in a language like C, because of the overhead of the VM and the lower degree of optimization possible. With programs using more complex data structures and control flows, this ratio can be as high as 50:1 [2]. Emscripten, a C-to-Javascript transpiler developed by the Mozilla Foundation, was created to address this issue and produce a highly-optimizable subset of statically typed, syntactically valid Javascript.

3. IMPLEMENTATION

Atlas makes progress on these issues with two interdependent extensions to Emscripten: a Javascript implementation of a subset of the OpenMP runtime and a server-client framework that distributes jobs and collects the results.

We begin by taking a C/C++ program and compiling it to asm.js using Emscripten, linking it with our custom OpenMP runtime. We execute the main process on the server, but upon the entering of a parallel section (marked

by a call to `__kmpc_fork_call`), the main process sends the current heap and source code along with the list of jobs to be run to the Atlas server, whose implementation will be discussed below. After performing some initial preprocessing, the server distributes these jobs to clients, who initialize their state and then jump to the outlined parallel section. When the job is complete, the client updates the state of the server using a "copy-on-write" model. These changes are merged on the server using the policy of "last write wins" (where order is determined by iteration index). When all jobs have completed and the results have been merged, the final diff is sent back to the main process, which then continues execution.

One of the major bottlenecks of this system is network latency. To minimize this, we distribute batches of jobs to clients and perform a client-side reduction before returning the results to the server. The unreliable nature of browsing sessions, however, is problematic – when a client leaves the page, the batch of jobs currently being executed must be re-executed. These batches must be intelligently sized to the predicted dwell time of the client in order to achieve optimal performance.

A YouTube dataset from a recent study at Cornell University was selected to train Atlas, which contained dwell time and sentiment analysis across a set of 1,125 randomly selected videos. A linear prediction model was constructed using the data, which provided a statistically significant R^2 value of 0.21. When a page containing the Atlas distribution framework loads, a request containing page metadata is asynchronously made to the server. That information is fed through the prediction model, and a set of jobs that totals the predicted dwell time are selected to be sent to the client.

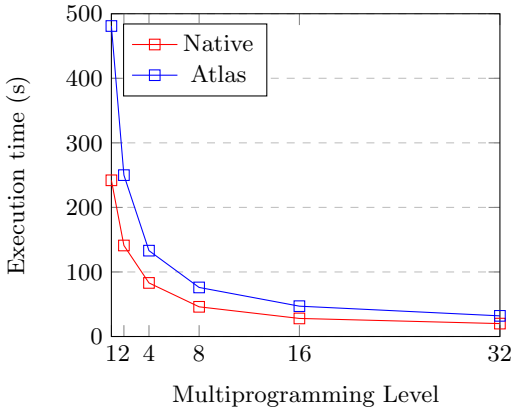
4. RESULTS

We benchmarked both the general performance of Atlas and the validity of our intelligent job distribution algorithm.

4.1 Performance on K-means Clustering

The highly parallelizable nature of K-means clustering makes it a good candidate for Atlas. We timed runs of the OpenMP and Javascript code from Andrea Ferretti's K-means Benchmark Project to compare the scalability of Atlas to that of a single-machine multicore system running the same code, as well as the raw performance of the compiled asm.js versus the idiomatic Javascript previous projects have sought to distribute.

30k iterations of k-means (Fig. 1)



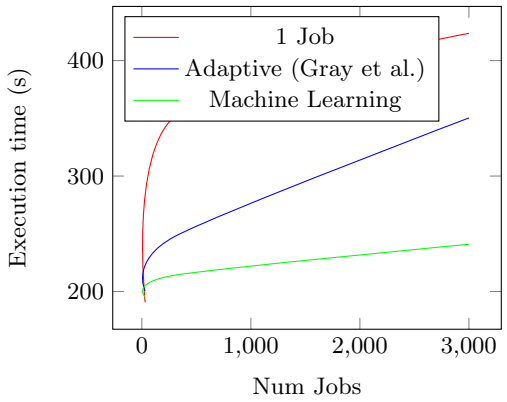
In Fig. 1, we see Atlas performs quite respectably – never less than half the speed of the equivalent native system, while demonstrating far greater scaling capacity.

Although native code running on multiple cores on a single machine will always be faster then a distributed system running Javascript on the same number of nodes, these results demonstrate that Atlas performs quite respectably – never less than half the speed of the equivalent native system, while demonstrating far greater capacity for scaling. Moreover, Atlas outperforms the pure Javascript implementation by a factor of over 20, demonstrating the power of asm.js as a tool for building high performance distributed systems. Had we had time to design and benchmark a distribution system for the pure Javascript code, this means its performance would have been capped at 5% of what we were able to achieve with this initial prototype of Atlas.

4.2 Performance of Intelligent Job Allocation Algorithm

The intelligent job sizing pipeline was tested using the Computer Language Benchmarks Game Spectral Norm benchmark, which computes the spectral norm of a matrix using the power method. For the purpose of benchmarking *Atlas*, the matrix was sized to 30,000. The results of this computation were compared against two other implementations, as described below.

Spectral norm with 30k matrix (Fig. 2)



In Fig.2, we see that the machine learning pipeline significantly outperforms both the naive approach and the adaptive algorithm proposed in the Gray system by a factor of 2x. This performance improvement stems from its approach to minimizing network requests. By intelligently calculating the amount of time it expects the browser to remain open, both inputs and results for multiple jobs can be sent via a single network request. This can be seen in the fact that all jobs require the same amount of computation – the only difference between the three algorithms is the number of calls to and from the server needed to receive inputs and return outputs.

5. ACKNOWLEDGEMENTS

Jason Brooks of Yale University contributed equally to this research and the manuscript.

6. REFERENCES

[1] Boldrin, F., Taddia, C., and Mazzini, G. Distributed computing through web browser. In *Vehicular Technology Conference, 2007.* (Sept 2007), pp. 2020–2024.

[2] Debian. The computer language benchmarks game, 2015.

[3] Martínez, G. J., and Val, L. Capataz: a framework for distributing algorithms via the World Wide Web. *CLEI Electronic Journal 18* (08 2015), 2 – 2.

[4] Pan, Y., White, J., Sun, Y., and Gray, J. Gray computing: an analysis of computing with background javascript tasks. In *Proceedings of the 37th International Conference on Software Engineering* (2015), IEEE Press.