A Robust Grid Computing Architecture on Heterogeneous Mobile Devices for Scientific Computing

By: Logan Garbarini

Introduction

Mobile computers in the form of cell phones, tablets, gaming consoles, and Personal Digital Assistants (PDAs) are becoming increasingly more powerful and are beginning to rival computers in computational power. Now, many people have several of these powerful devices. However, their hardware and software is vastly different from that of traditional general-purpose computers. If this power could be harnessed, a person's computational resources could be increased substantially (Büsching *et al.*, 2012). Furthermore, these same types of devices are starting to be used in more conventional roles. Currently, grid computing applications are designed to run on conventional thick OS platforms like Windows, Mac OS X, or Linux. By harnessing the power of the ever-increasing number of devices into a massively parallel architecture, small organizations or individuals could foreseeably create a fast heterogeneous compute cluster simply by using the multitude of devices at their disposal (Fadika *et al.*, 2012).

The technology for grid computing on mobile devices has not been established. One of the main problems is that programs such as BOINC and SETI@home are written in traditional programing languages for traditional architectures. They have not been designed for newer platforms. Furthermore, there has not been much interest because mobile devices have historically been much slower. However, there is a shift to more non-traditional architectures. For example, there are game consoles and small-form factor PCs that are Android-based. This opens the door for more advanced computing applications with more powerful processors. The rate of x86 chips sold has been decreasing whereas the number of SoC and ARM chips sold has

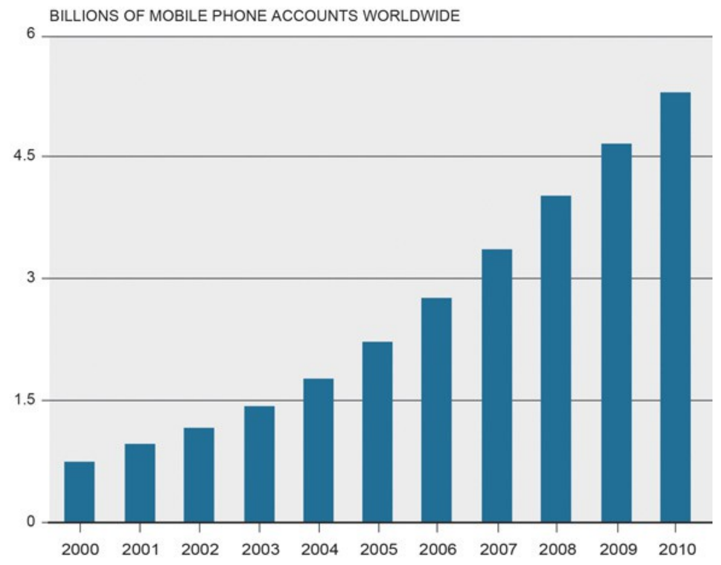

Figure 1: Mobile Phone Adoption Worldwide

Photo Credit: MIT Technology Review

been increasing over the past several years. This means that many distributed computing projects will be left high and dry when users opt for newer systems that cannot run traditional programs.

It is important that more powerful computer devices are developed because many of the conventional computer systems are getting retired. However, the amount of data scientists and businesses must deal with is increasing. Many scientific applications can even generate petabytes of data. Without a good grid computing architecture the power of newer devices will be limited to the power of a single device.

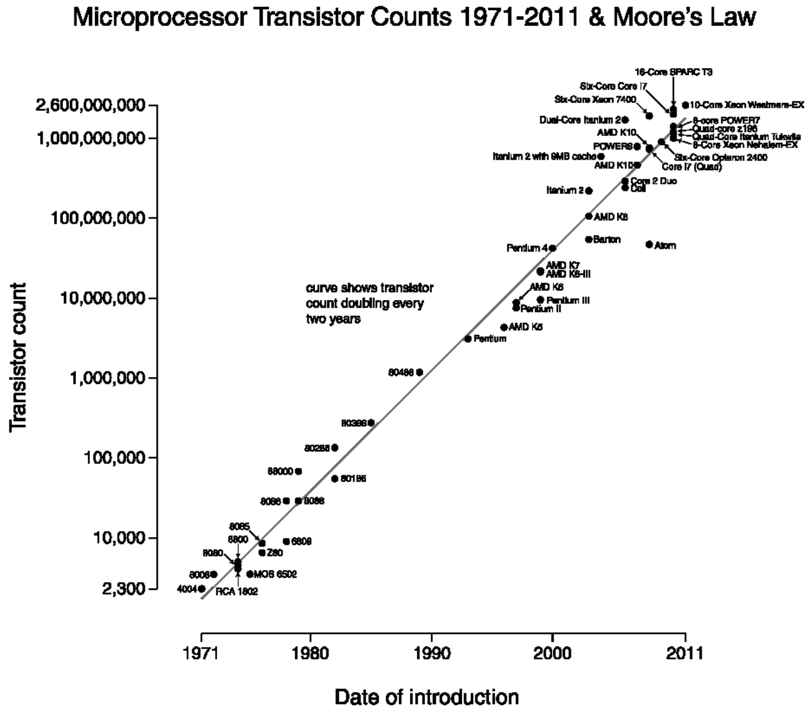

Figure 2: : Actual Transistor Count vs. Moore's Law

Photo Credit: Wikimedia Commons

Goals and Expected Outcomes

This investigation demonstrates a novel grid computing architecture that works on Android-based phones. Mobile devices have unique challenges compared to traditional computers. These devices suffer from battery constraints, unstable connections, slow connections speeds, and high latency interconnects (Chu and Humphrey 2004; Shah and Park 2011). Using purpose built software, this experiment minimizes those disadvantages and maximize the efficiency of the computational network.

The investigation helps determine the future utility of mobile grids. Based on the data gained, it will be possible to determine how effective the software is and whether there are any current real-world applications. The future uses of this technology will also be evaluated.

Methodology

This experiment used a combination of the Ruby on Rails Web Application Framework and the Java Programming language. These two pieces of software are used for the Server and the Worker systems respectively. The server is in charge of taking the tasks and splitting them into jobs which can be sent to the workers via HTTP in accordance with the REST (Representational State Transfer) principles. The workers will then process the jobs and return them through HTTP (Hyper Text Transfer Protocol).

Once all the workers have returned all the jobs, the server must process the Job data into the output for a job. The jobs were stored on a SQLite 3 database that was connected to Ruby on Rails server software. When a client sent a GET request, the job was retrieved and marked as “in progress” with a specific Time-to-Live(TTL). This time to live was about 10% greater than the expected time the task would take to complete on the slowest device. The “in progress” marker prevents the job from being assigned to a different worker and wasting the grid's time. If the client did not submit the completed job back within the specified TTL, the job was returned to the “Pending” state. If the client did complete the task before the TTL expires, the server would mark the task as “Completed”.

The grid also had a complex error handling system. If the client experienced an error or fails to complete the task in time, the error text would be submitted to the server and logged and the job would now have a failure count. If the failure count exceeded a threshold the job was marked as “failed” and would not be put back into the job pool. This ensured that even in the event of an anomalous job the cluster would not freeze the whole grid by assigning jobs that might be impossible (e.g. vertical asymptote in which the integral goes to infinity). Lastly, if a worker finishes a task past the TTL, the answer may still be submitted and the task marked as completed, but the failure count still remains. This allows only the suspect computations to be verified rather than the whole dataset.

This software architecture differs substantially from most other grid computing architectures in that it is stateless. Rather than keeping connections alive and eating through precious bandwidth and battery resources, this framework only sends data when necessary. This is hugely advantageous in comparison with many other current architectures which perform poorly when there is insufficient network resources. Additionally, REST allows the software to leverage many of the techniques used by mobile app developers to save on bandwidth including compression.

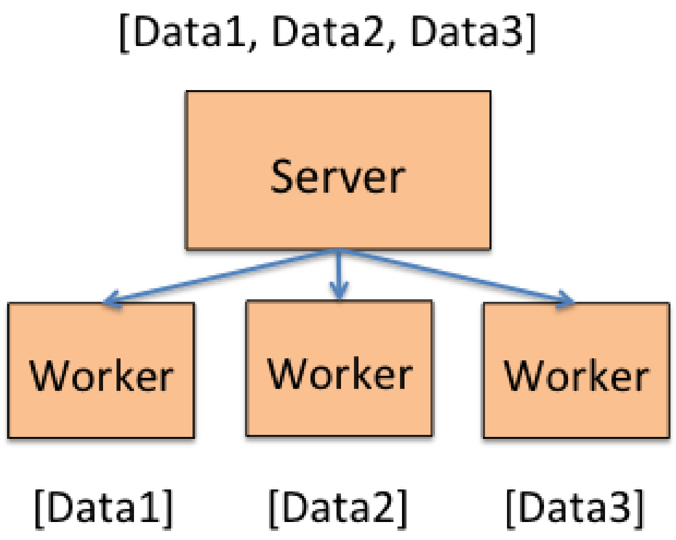

Illustration 1: This schematic diagram shows a high level architecture of the grid computing software.

To test this architecture, a heterogeneous set of Android based mobile devices were used to run a simple integration program in a parallel manner. The three devices used are the Motorola Triumph running Android 2.3, LG Nexus 4 running Android 4.2, and the Asus Nexus 7 running Android 4.2. Each of these devices will have the client software installed that will make requests at specified intervals (30s in this experiment). The workers then process the task and send it back to the server and request a new task. Additionally, the devices were placed at 20 feet from an 802.11bgn router that was being constantly used for web browsing. The device connection speed to the router was also monitored and varied from 48-72 Mbps (Megabits/sec). This simulated real-world conditions with a lightly congested network and average network performance.

A simple, easily parallelizable task was created that used Simpson's rule to conduct a numerical integration. A server-side script broke up the integration into jobs which were then added to the job pool. This task enables the simple testing of the grid without too much wasted time on the task itself. Although Simpson's rule may not be used very often for numerical integration, it provided a well understood task that was easy to troubleshoot and locate problems. The partition size *h* that will be used is 0.00001. Five different limits of integration were tested 0 to 100, 200, 300, 400 and 500.

$$\int \frac{e^{\sqrt{|sin(x)|}}}{\sqrt{2\pi}}$$

Equation 1: The function integrated using the Compute Grid and Simpson' Rule

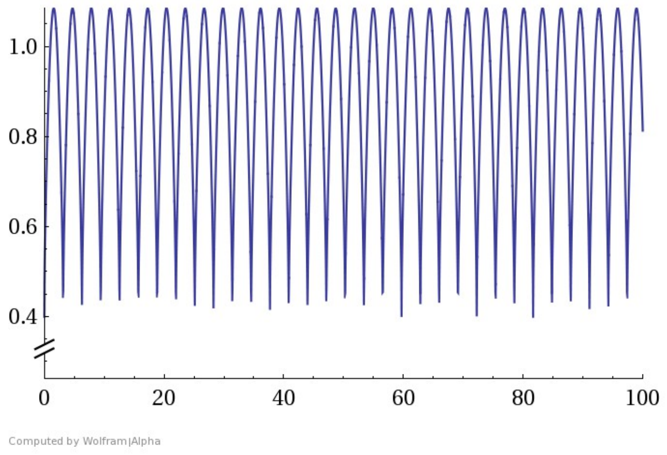

Figure 3: A graph showing the function being integrated
Graph computed by: WolframAlpha

$$\int_a^b f(x)\,dx \approx h\left[\frac{1}{3}f(x_0) + \frac{4}{3}f(x_1) + \frac{2}{3}f(x_2) + \cdots + \frac{4}{3}f(x_{n-1}) + \frac{1}{3}f(x_n)\right]$$

Equation 2: Simpson's rule was used to test the compute grid

Discussion and Results

Qualitative Data:

The system was extensively tested before taking quantitative data measurements. However, certain qualitative data was taken that would have been hard to quantify with the available resources:

- The error handling system ensured 99.9% of completable jobs were completed in every test with the most recent software version (over 40 task runs)
- Ten test runs showed the grid performed as expected when one of the jobs was designed to fail (e.g. infinite integral), that job failed and the rest were completed
- Several times during testing a worker became disconnected from the wireless network and reconnected. Although this sometimes resulted in the loss of the job data, the grid still completed the task demonstrating the resiliency of the network.

The observations demonstrate that the grid computing architecture is a stable program that is fully capable of handling various network loads, real-world conditions, and unexpected errors. Also by using real-world network conditions, the test is directly applicable to future applications.

Quantitative Data:

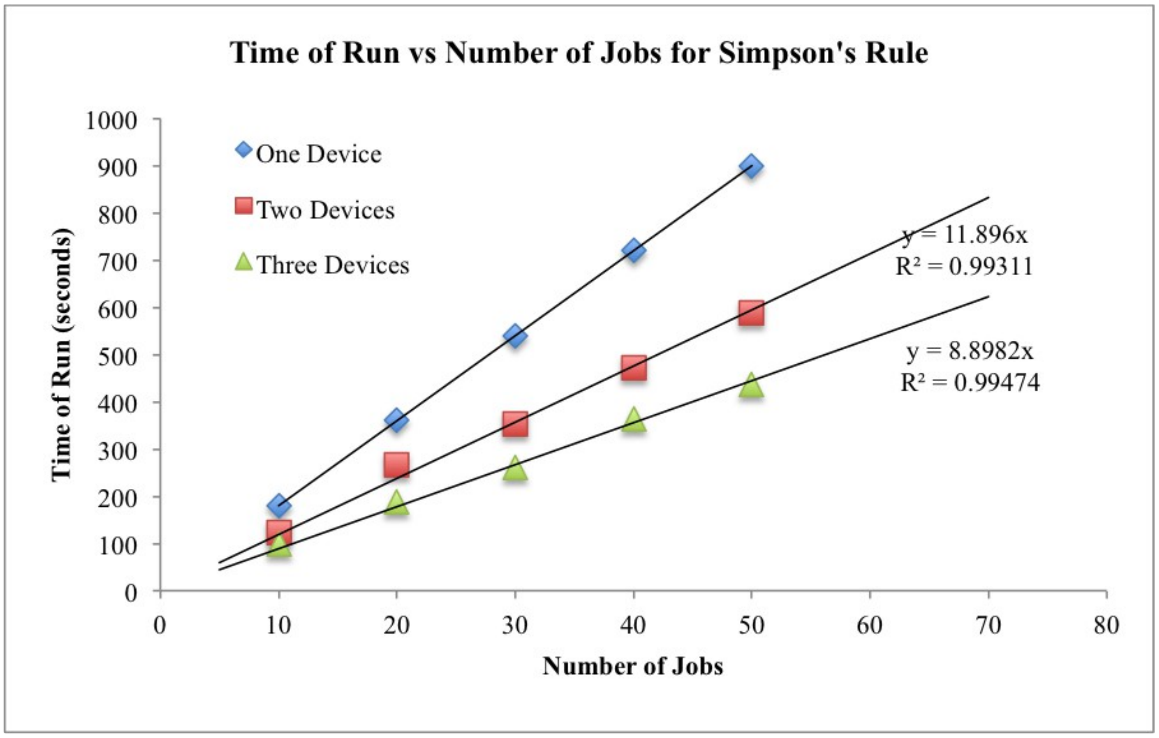

Figure 4: This graph shows how the number of workers has a larger effect on run time as the number of jobs increases. It also shows that the run time of jobs scales linearly with the number of jobs. The one device plot corresponds to the use of the Nexus 7, two devices corresponds to the Nexus 7 and the Motorola Triumph, and the three devices corresponds to the use of the Nexus 4, Nexus 7, and Motorola Triumph.

The data show that the compute cluster is capable of running the test computations and running through the data and returning a result. The grid is capable of handling progressively larger task sizes so that there will not be a noticeable degradation in speed. Also, the data show that the more workers, the faster the computation can run. The data show that the computer grid can be used to create a logical computer that can reach better computational speeds than could be achieved with a single device. This investigation shows that the software architecture and systems are capable of handling multiple devices allowing the server device to be freed up for other computations (single core CPU usage <5%). Once the computations were run, the data was plotted onto a graph that showed how the run time scaled linearly with the number of jobs. Also, the same graph shows the differences in the rate of computations between the different data runs. All three trials showed that the data had a direct linear relationship with $p < 0.02$.

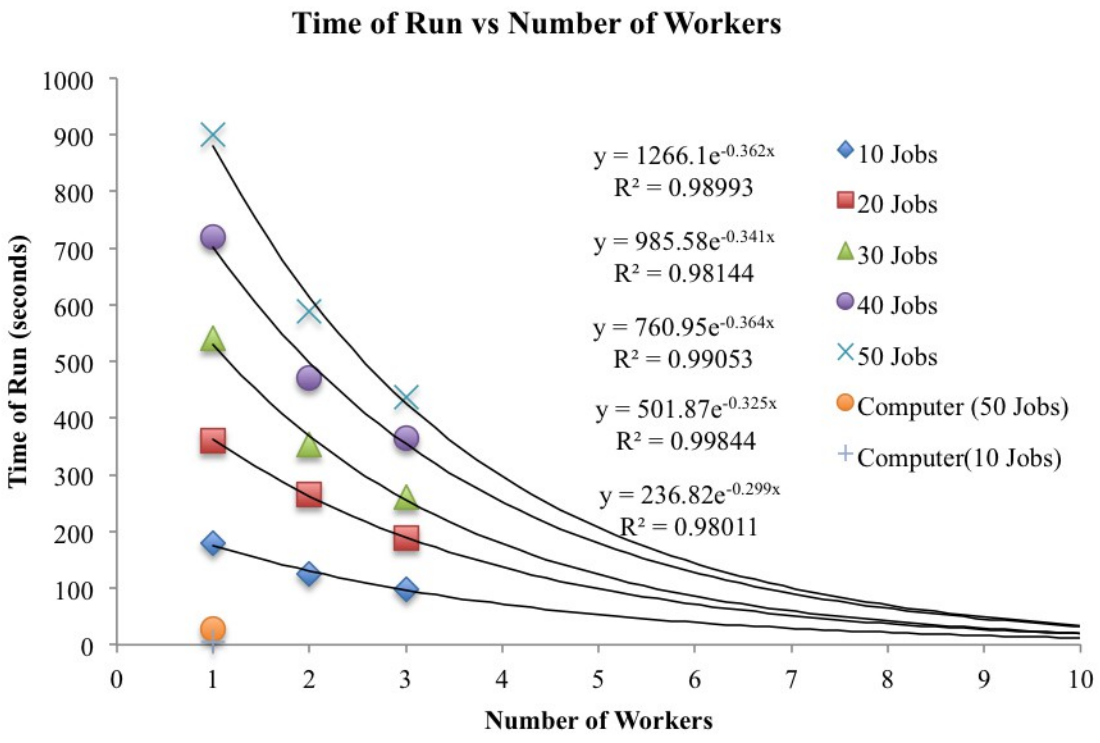

Figure 5: This graph shows how the number of workers affects the run time of the computation. Exponential regression was used for the trend lines. Worker 1 was the LG Nexus 7, Worker 2 was the Motorola Triumph, and Worker 3 was the Nexus 4. The exponential regression trendlines are used to show a projected extrapolation based on the theoretical model. The limits of this approach are highlighted in the discussion.

The data show how increasing the number of nodes could reduce the job run time. Unfortunately, it was not possible to use more devices because of budget constraints. Also, the Android Virtual Devices that had been planned on being used ran far too slowly to do any kind of real-world simulation. The data that was collected showed that there was a very high correlation ($p < 0.02$) when using an exponential regression. This relationship shows how more devices can increase the speed of the compute cluster.

Although the desktop computer is much faster at small values of N, when N is increased the trend shows that a compute cluster of about 14 devices will overtake a single computer and begin to reduce the computation time.

Conclusion

The ever improving speed and availability of inexpensive Android-based mobile devices will allow a new set of mobile grid applications that can harness the power of conventional computers as well as new mobile devices.

This investigation shows that grid computing on mobile phones is possible and provides a proof of concept architecture for running such a grid. This architecture shows that the processing can scale quite linearly and the size of the job does not factor in to the compute time.

The real-world nature of this investigation shows that the framework developed by the author is usable and is not just a simple laboratory curiosity. Additionally, observations show that under a wide variety of varying conditions the framework is robust. Even when errors do occur, the software is capable of logging the error without a complete failure of the entire compute grid.

The architecture also demonstrated that stateless HTTP requests are usable in high latency situations in grid computation. Using stateless requests is significantly more efficient in terms of network bandwidth and battery life and could prove useful not just in mobile grid computing applications but in general-purpose grid computation.

However, this investigation was limited by a lack of resources that prevented further investigation and limits the scope of the data. Had more devices been available, more data could have been collected and a better relationship demonstrated. However, the author tried to use available resources to develop a relationship by examining both the efficiency as the number of jobs as well as the efficiency as the number of devices were increased.

Additionally, the data support the conclusion that the compute power of the cluster can be easily increased to handle larger jobs. The data also show that with enough workers, the computational power of the grid could exceed that of a desktop computer. Additionally, it is clear that the ever increasing speed of mobile processors will enable mobile devices to start to perform more traditional loads in the near future.

Mobile grids offer a promising opportunity to improve computation speed using currently existing devices. While a few mobile devices could not rival a desktop or laptop, combining more of them together could yield a very powerful grid. Such grids would allow developing countries to pool resources to have more powerful computers than conventionally possible. It would also allow doctors or soldiers in the field to have access to vastly more computational resources than they might normally be able to achieve. As these devices become more and more powerful and are used in more conventional desktop environments, the applications of these grids will expand further.

Future Research

While this research does a decent job showing how mobile devices could help create a more powerful mobile device landscape, there is much more room for additional research. First, more devices could have been used to examine the information in more depth. Due to budget constraints it was only possible to test with three devices. This limited the ability to extrapolate the data and look for trends. Additionally, the lack of three identical devices limits the ability to process the data. However, all three devices are within an order of magnitude of single core performance. The LG Nexus 4 operated at about 50 Million Floating Point Operations per Second (MFLOPS), the Nexus 7 at about 45 MFLOPS, and the Motorola Triumph operated at around 40 MFLOPS when tested using a LINPACK benchmark(Linpack for Android). Initially, two additional devices had been planned to be used, however after examining their ability to run the LINPACK benchmark as well as the actual integration test, it was determined that the they were at least an order of magnitude too slow to use in the experiment.

Second, the application could be written to be multithreaded. During the experiment, it was observed that each multicore device was only using one core to do the computation. If the grid could be written to handle more tasks concurrently, the speed of the grid could have been substantially improved.

To improve the performance even further and to take advantage of mobile processing technology, GPU hardware acceleration could be employed to speed up certain types of computations. This would allow the grid to harness the very powerful graphics processors onboard many newer mobile devices. In the future, the author hopes to leverage the Android Native Development Kit (NDK) and use the RenderScript API which allows the programmer to execute computations on the GPU and at a lower level on the CPU. This could offer significant gains in performance over the high level CPU access provided by the Android Development Kit.

An additional improvement that could be made to improve the data would be to try different tasks on the grid to see if certain tasks run faster than others. For example, a Monte Carlo Simulation could be run or image processing techniques could be used. Certain tasks leverage different functions that may perform vastly different on certain hardware and software architectures. Further research might find that certain types of tasks are ideally suited for mobile grid computing.

Lastly, although the resilience of the grid computing architecture was demonstrated, it would be worth comparing current message passing systems to the RESTful transfers used here on different network topographies. The author has already started looking into network modeling and hopes to be able to compare the resilience of different conventional as well as novel architectures.