MapReduce System over Heterogeneous Mobile Devices

Peter R. Elespuru, Sagun Shakya, and Shivakant Mishra

Department of Computer Science
University of Colorado, Campus Box 0430
Boulder, CO 80309-0430, USA

Abstract. MapReduce is a distributed processing algorithm which breaks up large problem sets into small pieces, such that a large cluster of computers can work on those small pieces in an efficient, timely manner. MapReduce was created and popularized by Google, and is widely used as a means of processing large amounts of textual data for the purpose of indexing it for search later on. This paper examines the feasibility of using smart mobile devices in a MapReduce system by exploring several areas, including quantifying the contribution they make to computation throughput, end-user participation, power consumption, and security. The proposed MapReduce System over Heterogeneous Mobile Devices consists of three key components: a server component that coordinates and aggregates results, a mobile device client for iPhone, and a traditional client for reference and to obtain baseline data. A prototypical research implementation demonstrates that it is indeed feasible to leverage smart mobile devices in heterogeneous MapReduce systems, provided certain conditions are understood and accepted. MapReduce systems could see sizable gains of processing throughput by incorporating as many mobile devices as possible in such a heterogeneous environment. Considering the massive number of such devices available and in active use today, this is a reasonably attainable goal and represents an exciting area of study. This paper introduces relevant background material, discusses related work, describes the proposed system, explains obtained results, and finally, discusses topics for further research in this area.

Keywords: MapReduce, iPhone, Android, Mobile Platforms, Apache, Ruby, PHP, jQuery, JavaScript, AJAX.

1 Introduction

Distributed computing has come into its own in the internet age. Such a large computational pool has given rise to endeavors such as the SETI@Home [14], and Folding@Home [7] projects, which both attempt to allow any willing person to surrender a portion of their desktop computer or laptop to a much larger computational goal. In the case of SETI@Home, millions of users participate to analyze data in search of extra terrestrial signals therein, whereas Folding@Home

S. Lee and P. Narasimhan (Eds.): SEUS 2009, LNCS 5860, pp. 168–179, 2009.
© IFIP International Federation for Information Processing 2009

is a bit more practical. Folding@Home's goal is "to understand protein folding, misfolding, and related diseases". These systems, along with others which are mentioned later, are conceptually similar to what we propose, which is a system that allows people to participate freely in these kinds of massive, computationally bound problems so that results may be quickly obtained. There are many similar approaches to solving large computationally intensive problems. One of the most famous of these is the problem of providing relevant search of the Internet itself [2]. Google has emerged as the superior provider of that capability, and a portion of that superiority comes by way of the underlying algorithms in use to make their process efficient, elegant, and reliable [13], MapReduce [4]. MapReduce is similar to other mechanisms employing parallel computations, such as parallel prefix schemes [12] and scan primitives [3], and is even fairly similar to blocked-sort based indexing algorithms [16].

We believe there exists a blatant disregard of certain capable devices [11] in the context of these kinds of distributed systems. Existing implementations have neglected the mobile device computation pool, and we suspect this is due to a number of factors which hamper most current mobile devices. It seems only smart phones are powerful enough, computation wise, for most of these distributed workloads. There are many additional concerns as well that have been covered by prior work, such as power usage, security concerns [9] and potential interference with the device's intended usage model as a phone. All of these factors limit the viability of incorporating mobile devices into a distributed system. It is our belief that despite these limitations, there are solutions that allow the inclusion of the massive smart phone population [6] into a distributed system. One logical progression of MapReduce, and other such distributed algorithms, is toward smart mobile devices primarily because there are so many of them, and they are largely untapped. Even a small scale incorporation of this class of device can have an enormous impact on the systems at large and how they accomplish their goals. Increases in data volume underscores the need for additional computational power as the world continues to create far more data than it can realistically and meaningfully process [5]. Using smart mobile devices, in addition to the more traditional set of servers, is one possible way to increase computational power for these kinds of systems, and is exactly what we attempt to prove and quantify in specific cases by leveraging prior work on MapReduce.

This paper explores the feasibility of using smart mobile devices in a MapReduce system by exploring several areas, including quantifying the contribution they make to overall computation throughput, end-user participation, power consumption, and security. We have implemented and experimented with a prototype of a MapReduce system that incorporates three types of devices: a standard Linux server, an iPhone, and an iPhone simulator. Preliminary results from our performance measurements support our claim that mobile devices can indeed contribute positively in a large heterogenous MapReduce system, as well as similar systems. Given that the number of smart phones is clearly on the rise, there is immense potential in using them to build computationally-intensive parallel processing applications.

The rest of the paper is organized as follows. In Section 2, we briefly outline the MapReduce system. Section 3 touches on similar endeavors. In Section 4, we describe the design of our system, and in Section 5, we describe the implementation details. In Section 7, we discuss experimental results measured from our prototype implementation. Next, we discuss some optimizations in Section 8 and then finally conclude our paper in Section 10.

2 MapReduce

MapReduce [4] is an increasingly popular programming paradigm for distributed data processing, above and beyond merely indexing text. At the highest architectural level, MapReduce is comprised of a few critical pieces and processes. If you have a large collection of documents or text, that corpus must be broken into manageable pieces, called splits. Commonly, a split is one line of a document, which is the model we follow as well. Once split, a master node must assign splits to workers who then process each piece, store some aspect of it locally, but ultimately return it to the master node or something else for reduction. The reduction then typically partitions the results for faster usage, accounting for statistics, document identification and so on.

We describe the MapReduce process in three phases: Map, Emit, and Reduce (See Figure 1). In our system, the map phase is responsible for taking a large data set, and chunking it into splits. The Emit phase is entails the distributed processing nodes obtaining and work on the splits, and returning a processed result to another entity, the master node or job server that coordinates everything. Unlike most MapReduce implementations, the nature of mobile devices precludes us from using anything other than network communications to read and write data, as well as assign jobs and process them. The final phase is Reduce, which in our case further minimizes the received results into a unique set of data that ultimately gets stored in a database for simplicity.

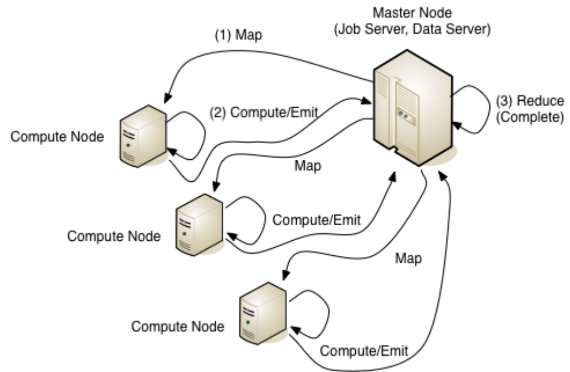

Fig. 1. High Level Map Reduce Explanation

For example, given a large set of plain text files over which you may wish to search by keyword, a MapReduce system begins with a master node that takes all those text files, splits them up line by line, and parcels them out to participants. The participating computation nodes find the unique set of keywords in each line of text they were given, and emit that set back to the master node. The master node, after getting all of the pieces back, aggregates all

of the responses to determine the overall unique set of keywords for that whole set of data, and stores the result in a database, file, or some other persistent storage medium. It is at this point the data is analyzed and searched whatever way desired from within our web application. One of the biggest strengths of MapReduce lies in its inherent distribution of phases, which results in an extremely high degree of reliable parallelism when implemented properly. MapReduce is both fault and slow-response tolerant, which are very desirable characteristics in any large distributed system.

3 Related Work

There have been a number of other explorations of heterogeneous MapReduce implementations and their performance [15], as well as some more unique expansions on the idea such as using JavaScript in an entirely client side browser processing framework [8] for MapReduce. None of this related work however focuses on using a mobile device pool as a major computation component. To complement these related works, we focus on mobile devices, and in particular, on the specifics of heterogeneity in the context of mobile devices mixed with more traditional computation resources.

4 The Heterogeneous Mobile Device MapReduce System

Our problem encompasses three areas: 1) Provide a mechanism for interested parties to participate in a smart phone distributed computational system, and ensure they are aware of the potential side effects; 2) Make use of this opt-in device pool to compute something and provide aggregate results; and 3) Provide meaningful results to interested parties, and summarize them in a timely fashion, considering the reliability of devices on wireless and cellular networks. Our solution is The Heterogeneous Mobile Device MapReduce System.

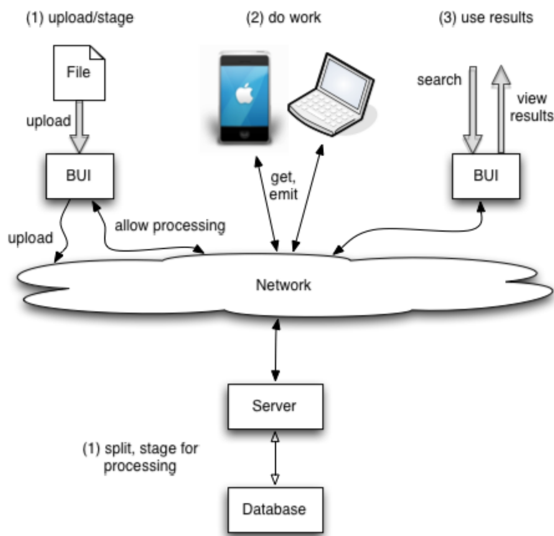

Fig. 2. System Summary

There are several key components in our system: 1) A server which acts as the master node and coordinator for MapReduce processing; 2) Server side client code used to provide faster more powerful client processing in conjunction with mobile devices, 3) The mobile device client which

implements MapReduce code to get, work on, and emit results of data from the master node; and finally 4) The BUI, or browser user interface (web application), which lets the results be searched (See Figure 2).

The MapReduce master node server leverages the Apache [17] web server for HTTP. To provide the MapReduce stack, we actually have two different implementations of our master node/job server code, one in Ruby [18] and one in PHP [19]. However, we primarily used the PHP variant during our testing. Once the master node has been seeded with some content to process, it is told to begin accepting participant connections. Once the process begins, clients of any type, mobile or traditional, may connect, get work, compute and return results. During processing, clients, whether they are mobile devices or processes running on a powerful server, can continually request work and compute results until nothing is left to do for a given collection. In this case, the server still responds to requests, but does not return work units since the cycle is complete (See Figure 3).

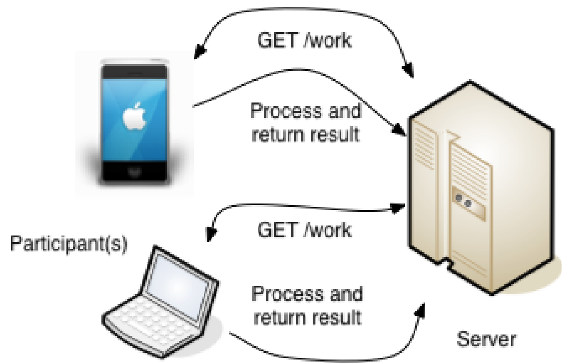

Fig. 3. Client Flow

After all the data has been processed, clients can still request work, but obviously are not furnished anything. At this point, our web application front end is used to search for keywords throughout the documents which were just processed. The web application was implemented in PHP and makes use of the jQuery [20] JavaScript framework to provide asynchronous (AJAX) page updates as workers complete units, in real-time. More can be seen in Figure 2. Further, Figure 4 illustrates exactly what the entire process looks like.

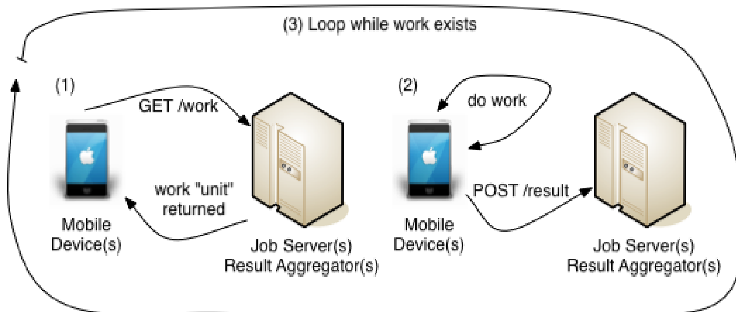

Fig. 4. Work Loop

5 System Development

There are a few additional aspects of developing this system that warrant discussion. Our experience with the development environment, and lessons learned are worth sharing as well.

5.1 Mobile Client Application Development Experience

We developed our mobile client application on the iPhone OS platform using the iPhone SDK, Cocoa Touch framework and Objective-C programming language. As part of the iPhone SDK, the XCode development environment was used for project management, source code editing and debugging. To run and test the MapReduce mobile client, we used the iPhone simulator in addition to actual devices. Apple's Interface Builder provided a drag and drop tool to develop the user interface very rapidly. All in all the experience was extremely positive [10].

5.2 Event Driven Interruption on iPhone

Event handling on the iPhone client proved rather interesting, due largely to the fact that certain events can override an application and take control of the device against an application's will. While the iPhone is processing data, other events like an incoming phone call, a SMS message or a calendar alert event can take control of the device. In the case of an incoming phone call, the application is paused. Once the user hangs up, the iPhone client is relaunched by iPhone OS, but it is up to the application to maintain state. While on the call, if the user goes back to the home screen or launches another application, the phone client does not resume, and again the application is responsible for maintaining state. When an SMS message or calendar event occurs the computation continues in the background unless the user clicks on the message or views the calendar dialog. In the latter case the action is same as when there is phone call. These events, which are entirely out of the control of the application, pose an interesting challenge and must be addressed during development.

6 End-User Participation

Participants are largely in two different camps, captive and voluntary. For example, if a system such as ours was deployed in a large corporation where most employees have company provided mobile devices, that company could require employees to allow their devices to participate in the system. These are what we consider captive users. Normal users on the other hand are true volunteers, and participate for different reasons. The key is to come up with methods which engage both of these types of users so that the overall experience is positive for everyone involved.

There are a large number of possible solutions to entice both types of users. Both captive and voluntary users could be offered prizes for participation, or perhaps simply receive accolades for being the participant with the most computed work units. This is similar to what both SETI@Home and Folding@Home

do, and has proven effective. The sense of competition and participation drives people to team up with the hopes of being the most productive participant. This topic is discussed further later on as well.

7 Results

Our results were very interesting. We created several data sets of varying sizes composed of randomly generated text files of varying sizes. Data set sizes overall ranged from 5 MB to almost 50 MB. Within those data sets, each individual text document ranged in size as well, from a few kilobytes up to roughly 64 kilobytes each. Processing throughput was largely consistent independent of both the overall data set size and the distribution of included document sizes.

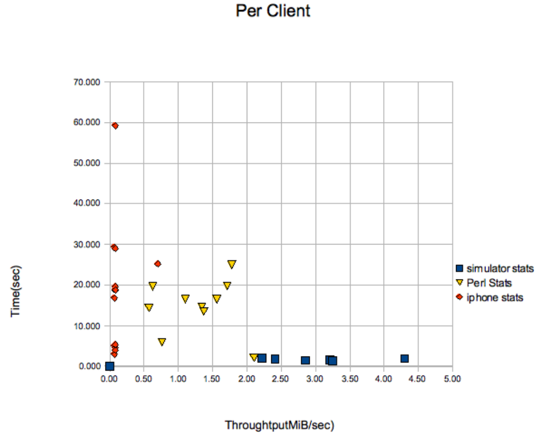

Fig. 5. Client Type Comparison

Figure 5 illustrates exactly what we expected would be the case. The simulated iPhone clients were the fastest, followed by the traditional perl clients, and lastly the real iPhone clients, which processed data at the slowest rate of all clients tested. The reason this behavior was expected is that the simulated iPhone clients ran on the same machine as the server software during our tests. The perl clients were executed on remote Linux machines. Interestingly though, mixing and matching client types didn't seem to impact the contribution of any one particular client type. Perl clients processed data at roughly the same rate independent of whether a given test included only perl clients, as did simulated and real iPhone clients.

Figure 6, presents another visualization that clearly shows there was a fair amount of variation in the different client types. Again, the simulated iPhone clients were able to process the most data, primarily because they were run on the same machine as the server component. The traditional perl clients were not far behind, and the real iPhone clients were the laggards of the bunch.

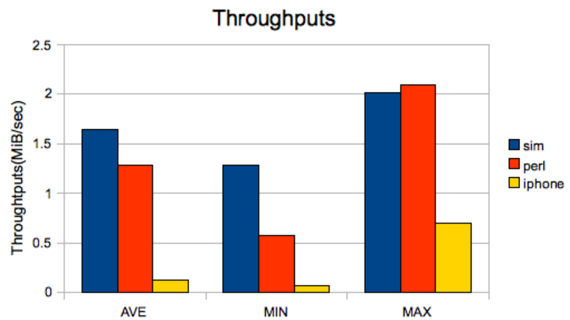

Fig. 6. Min Max Average

7.1 Interpretation of the Results

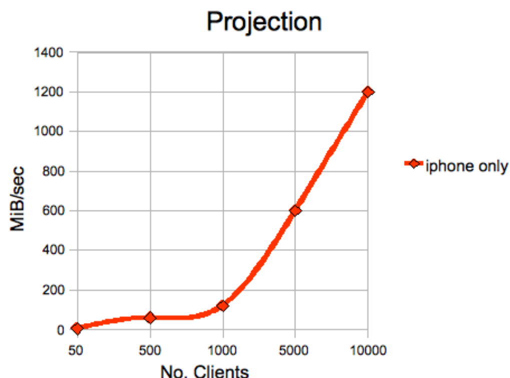

Fig. 7. Projected System Throughput

Simulated iPhone clients processed an average of 1.64 MB/sec, Perl clients processed an average of 1.29 MB/sec, and finally, real iPhone clients processed an average of 0.12 MB/sec. The simulated iPhone clients can be thought of as another form of local client, and they help highlight the difference and overhead in the wireless connection and processing capabilities of the real phones. These results and averages were consistent across a variety of data sets, both in terms of size and textual content. Our results show that very consistently, the iPhones were capable of performing at roughly an order of magnitude slower than the traditional clients, which is a very exciting result. It implies that a large portion of processing could be moved to these kinds of mobile clients, if enough exist at a given time to perform the necessary work load. For example, a company could purchase one server to operate as the master node, and farm all of the processing to mobile devices within their own company. Provided they have on the order of one hundred or more employees with such devices, which is a very likely scenario. This also suggests that this system could be particularly useful for non-time sensitive computations. For example, if a company had a large set of text documents it needed processed, it could install a client on its employees mobile devices. Those devices in turn could connect and work on the data set over a long period of time, so long as they are capable of processing data faster than it is being created. Considering how easy it is to quantify the contribution each device type is capable of making, such a system could very easily monitor its own progress. In summary, there are a large number of problems to which this system is a viable and exciting solution.

We had a limited number of actual devices to test with (3 to be specific) but all performed consistently across all tests and data sets, so we feel comfortable projecting forward to estimate the impact of even more devices. As you increase the number of actual devices, throughput should grow similarly to what is represented in Figure 7. If a system utilized 500 mobile devices, we expect that system would be capable of processing close to 60 MB/sec of textual data. Similarly, 10000 devices would likely yield the ability to process 1,200 MB/sec (1.2 GB/sec!) of data. This certainly suggests our system warrants further exploration, but points to the fact that other components of the system would definitely start becoming bottlenecks. For example, at those rates, it would take massive network bandwidth to even support the data transfer necessary for the processing to take place.

8 Optimizations

There are a few areas where certain aspects of this system could be improved to provide a more automatic and ideal experience. It is particularly important that the end user experience be as automatic and elegant as possible.

8.1 Automatic Discovery

Currently, the client needs to know the IP address and port number of the server in order to participate. This requires prior knowledge of the server address information which may be a barrier to entry for our implementation of MapReduce. In order to allow auto-discovery, we can have Bonjour (aka mDNS), a service discovery protocol, running on the server and clients. Bonjour automatically broadcasts the service being offered for use. With Bonjour enabled on the server and clients a WiFi network is not an absolute requirement. However, there are some limitations of using Bonjour as service discovery protocol. Namely, all devices must be on the same subnet of the same local area network, which imposes maximum client limits that would minimize the viability of our system in those situations.

8.2 Device Specific Scaling

An important goal of this system is that it can be used on heterogeneous mobile devices. As such, not all mobile devices perform the same or have the same power usage characteristics. The system should ideally have the ability to know about each type of device it can run on to maintain a profile of sorts. The purpose is to allow the system to optimize itself. For example, on a Google Android device it would have one profile, and on an iPhone it would have another, and in each case the client application would taylor itself to the environment on which it is running. The ultimate goal is to maximize performance relative to power consumption.

8.3 Other Client Types

In addition to smart mobile devices of various types, and traditional clients, there are other kinds of clients which could be used in conjunction with the other two varieties. In particular, a JavaScript client would allow any web browser to connect and participate [8] in the system. The combination of these three types of clients would be formidable indeed, and form a potentially massive computational system.

9 Additional Considerations

There are a number of areas we did not explore as part of our implementation. However, the following topics would need to be considered in an actual production implementation.

9.1 Security

Security is a multi-faceted topic in this context. Our primary concerns are two fold, first can the client implementation impact security of end-user's mobile devices, or in any way be used to compromise their devices. Second, is the data mobile devices receive in the course of participating information that would be risky to have out in the open, were a device compromised by some other means. A production implementation would need to ensure that even if a mobile device is compromised by some other means, any data associated with this processing system is inaccessible. One way to accomplish this might be to improve the client to store its local results in encrypted form, and transmit them via HTTPS to ensure the communication channel also minimizes the opportunity for compromise. Another consideration that must be made is whether to process sensitive information in the first place. In fact, certain regulations may even prevent it altogether.

9.2 Power Usage

Power usage is a very critical topic when considering a system such as this. The additional load placed on mobile devices will certainly draw more power, which is potentially even disastrous in some situations. For example, if the mobile device is an emergency phone, running down the battery to participate in a computation cycle is a very bad idea. Ultimately, power usage must be considered when deciding which devices to allow in the mix. There are a number of things which may be done to account for these concerns, such as adding code to the mobile client that would prevent it from participating if a certain power level has been passed. This may prove particularly tricky however, since not all mobile platforms include API calls that allow an application to probe for that kind of low level system information. A balance must be reached, and it is the responsibility of the client application implementation to ensure that balance is maintained.

9.3 Participation Incentives

Regardless of whether a participating end user is a captive corporate user or a truly voluntary end user, there should be an incentive structure that rewards participation in a manner that benefits all parties. There are several incentives that could be considered. One potential way would be to offer some kind of reward for participating, based on the number of work units completed for example. This would entice users to participate even more. A community or marketplace could even be set up around this concept. For example, companies could post documents they want processed and offer to pay some amount per work unit completed. Users could agree to participate for a given company, and allow their device to churn out results as quickly as possible. This would have to be a small amount of money per work unit to be viable, perhaps a few cents each. Such a marketplace could easily become quite large and be very beneficial to all involved. Amazon has a similar concept in place with its Mechanical Turk, that

allows people to post work units which other people then work on for a small sum of money [1]. Another possibility would be to bundle the processing into applications where it runs in the background, such as a music player, so that work goes on continually while the media player is playing music. The incentive could be a discount of a few cents when purchasing songs through that application, relative to some number of successfully completed jobs. The possibilities are numerous.

10 Conclusions

As is clearly evident in our results, mobile devices can certainly contribute positively in a large heterogenous MapReduce system. The typical increase from even a few tens of mobile devices is substantial, and will only increase as more and more mobile devices participate. Assuming a good server implementation exists, the mobile client contribution should increase with each new mobile device added. It is expected there would be a point of diminishing returns relative to network communication overhead, but the potential benefit is still very real. If non-captive user bases could be properly motivated, there is a large potential here to process massive amounts of data for a wide range of uses. This is conceptually similar to existing cloud computing, but where computation and storage resources happen to be mobile devices, or they interoperate between the traditional cloud and a new set of mobile cloud resources.

References

1. Amazon, Inc. Amazon Mechanical Turk, `https://www.mturk.com/mturk/welcome`
2. Barroso, L.: Web Search for a Planet: The Google Cluster Architecture. IEEE 23(2) (March 2003)
3. Blelloch, G.E.: Scans as Primitive Parallel Operations. IEEE Transactions on Computers 38(11) (November 1989)
4. Dean, J., Ghemawat, J.: Map Reduce, Simplied Data Processing On Large Clusters. ACM, New York (2004)
5. Dubey, P.: Recognition, Mining, and Synthesis Moves Computers to the Era of Tera. Technology@Intel Magazine (February 2005)
6. Egha, G.: Worldwide Smartphone Sales Analysis, UK (February 2008)
7. Folding@Home. Folding@Home project, `http://folding.stanford.edu/`
8. Grigorik, I.: Collaborative MapReduce in the Browser (2008)
9. Hunkins, J.: Will Smartphones be the Next Security Challenge (October 2008)
10. iPhone Developer Program. iphone development, `http://developer.apple.com/iphone/program/develop.html`
11. Krazit, T.: Smartphones Will Soon Turn Computing on its Head, CNet (March 2008)
12. Ladner, R.E., Fischer, M.J.: Parallel Prex Computation. Journal of the ACM 27(4) (October 1980)
13. Mitra, S.: Robust System Design with Built-in Soft-Error Resilience. IEEE 38(2) (February 2005)

14. SETI@Home. SETI@Home Project, http://setiathome.ssl.berkeley.edu/
15. Zaharia, M., Konwinski, A., Joseph, A.D., Katz, R., Stoica, I.: Improving MapReduce Performance in Heterogeneous Environments. In: OSDI (2008)
16. Manning, C., Prabhakar, R., Hinrich, S.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
17. Apache Web Server. Apache, http://httpd.apache.org/
18. Ruby Programming Language Ruby, http://www.ruby-lang.org/en/
19. PHP Programming Language PHP, http://www.php.net/
20. jQuery JavaScript Framework. jQuery, http://jquery.com/