Data-Intensive Supercomputing: The case for DISC

Randal E. Bryant

May 10, 2007
CMU-CS-07-128

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Google and its competitors have created a new class of large-scale computer systems to support Internet search. These "Data-Intensive Super Computing" (DISC) systems differ from conventional supercomputers in their focus on data: they acquire and maintain continually changing data sets, in addition to performing large-scale computations over the data. With the massive amounts of data arising from such diverse sources as telescope imagery, medical records, online transaction records, and web pages, DISC systems have the potential to achieve major advances in science, health care, business efficiencies, and information access. DISC opens up many important research topics in system design, resource management, programming models, parallel algorithms, and applications. By engaging the academic research community in these issues, we can more systematically and in a more open forum explore fundamental aspects of a societally important style of computing.

Keywords: parallel computing, data storage, web search

When a teenage boy wants to find information about his idol by using Google with the search query "Britney Spears," he unleashes the power of several hundred processors operating on a data set of over 200 terabytes. Why then can't a scientist seeking a cure for cancer invoke large amounts of computation over a terabyte-sized database of DNA microarray data at the click of a button?

Recent papers on parallel programming by researchers at Google [13] and Microsoft [19] present the results of using up to 1800 processors to perform computations accessing up to 10 terabytes of data. How can university researchers demonstrate the credibility of their work without having comparable computing facilities available?

1 Background

This document describes an evolving set of ideas about a new form of high-performance computing facility that places emphasis on *data*, rather than raw computation, as the core focus of the system. The system is responsible for the acquisition, updating, sharing, and archiving of the data, and it supports sophisticated forms of computation over its data. We refer to such such system as *Data-Intensive Super Computer* systems, or "DISC." We believe that DISC systems could yield breakthroughs in a number of scientific disciplines and other problems of societal importance.

Much of our inspiration for DISC comes from the server infrastructures that have been developed to support search over the worldwide web. Google and its competitors have created DISC systems providing very high levels of search quality, response time, and availability. In providing this service, they have created highly profitable businesses, enabling them to build ever larger and more powerful systems. We believe the style of computing that has evolved to support web search can be generalized to encompass a much wider set of applications, and that such systems should be designed and constructed for use by the larger research community.

DISC opens up many important research topics in system design, resource management, programming models, parallel algorithms, and applications. By engaging the academic research community in these issues, we can more systematically and in a more open forum explore fundamental aspects of a societally important style of computing. University faculty members can also incorporate the ideas behind DISC into their courses, ensuring that students will learn material that will be increasingly important to the IT industry.

Others have observed that scientific research increasingly relies on computing over large data sets, sometimes referring to this as "e-Science" [18]. Our claim is that these applications call for a new form of system design, where storage and computation are colocated, and the systems are designed, programmed, and operated to enable users to interactively invoke different forms of computation over large-scale data sets. In addition, our entire world has been increasingly data intensive, as sensor, networking, and storage technology makes it possible to collect and store information ranging from retail transactions to medical imagery. DISC systems will enable us to create new efficiencies and new capabilities well beyond those already achieved by today's information technology.

This paper outlines the case for DISC as an important direction for large-scale computing systems. It also argues for the need to create university research and educational projects on the design, programming, and applications of such systems. Many people have contributed and helped refine the ideas presented here, as acknowledged in Appendix A.

1.1 Motivation

The following applications come from very different fields, but they share in common the central role of data in their computation:

- *Web search without language barriers.* The user can type a query in any (human) language. The engine retrieves relevant documents from across the worldwide web in all languages and translates them into the user's preferred language. Key to the translation is the creation of sophisticated statistical language models and translation algorithms. The language model must be continuously updated by crawling the web for new and modified documents and recomputing the model. By this means, the translation engine will be updated to track newly created word patterns and idioms, such as "improvised explosive devices."

  Google already demonstrated the value of applying massive amounts of computation to language translation in the 2005 NIST machine translation competition. They won all four categories of the competition in the first year they entered, translating Arabic to English and Chinese to English [1]. Their approach was purely statistical. They trained their program using, among other things, multilingual United Nations documents comprising over 200 billion words, as well as English-language documents comprising over one trillion words. No one in their machine translation group knew either Chinese or Arabic. During the competition, they applied the collective power of 1000 processors to perform the translations.

- *Inferring biological function from genomic sequences.* Increasingly, computational biology involves comparing genomic data from different species and from different organisms of the same species to determine how information is encoded in DNA. Ever larger data sets are being collected as new sequences are discovered, and new forms of derived data are computed. The National Center for Biotechnology Innovation maintains the GenBank database of nucleotide sequences, which has been doubling in size every 10 months. As of August, 2006, it contained over 65 billion nucleotide bases from more than 100,000 distinct organisms.

  Although the total volume of data is less than one terabyte, the computations performed are very demanding. In addition, the amount of genetic information available to researchers will increase rapidly once it becomes feasible to sequence the DNA of individual organisms, for example to enable *pharmacogenomics*, predicting a patient's response to different drugs based on his or her genetic makeup.

- *Predicting and modeling the effects of earthquakes.* Scientists are creating increasingly detailed and accurate finite-element meshes representing the geological properties of the earth's crust, enabling them to model the effect of a geological disturbance and the probabilities of earthquakes occurring in different regions of the world [3]. These models are continually updated as actual earthquake data are analyzed and as more sophisticated modeling techniques are devised. The models are an important shared resource among geologists, computational scientists, and civil engineers.

- *Discovering new astronomical phenomena from telescope imagery data.* Massive amounts of imagery data are collected daily, and additional results are derived from computation applied to that data [26]. Providing this information in the form of a shared global database would reduce the redundant storage and computation required to maintain separate copies.

- *Synthesizing realistic graphic animations.* The system stores large amounts of motion capture data and uses this to generate high quality animations [23]. Over time, the motion data can be expanded and refined by capturing more subjects performing more tasks, yielding richer and more realistic animations.

- *Understanding the spatial and temporal patterns of brain behavior based on MRI data.* Information from multiple data sets, measured on different subjects and at different time periods, can be jointly analyzed to better understand how brains function [22]. This data must be updated regularly as new measurements are made.

These and many other tasks have the properties that they involve collecting and maintaining very large data sets and applying vast amounts of computational power to the data. An increasing number of *data-intensive* computational problems are arising as technology for capturing and storing data becomes more widespread and economical, and as the web provides a mechanism to retrieve data from all around the world. Quite importantly, the relevant data sets are not static. They must be updated on a regular basis, and new data derived from computations over the raw information should be updated and saved.

2 Data-Intensive Super Computing

We believe that these data-intensive computations call for a new class of machine we term a *data-intensive super computing* system, abbreviated "DISC". A DISC system is a form of supercomputer, but it incorporates a fundamentally different set of principles than mainstream supercomputers. Current supercomputers are evaluated largely on the number of arithmetic operations they can supply each second to the application programs. This is a useful metric for problems that require large amounts of computation over highly structured data, but it also creates misguided priorities in the way these machines are designed, programmed, and operated. The hardware designers pack as much arithmetic hardware into the systems they can, disregarding the importance of incorporating computation-proximate, fast-access data storage, and at the same time creating machines that are very difficult to program effectively. The programs must be written in terms of very low-level primitives, and the range of computational styles is restricted by the system structure. The systems are operated using a batch-processing style, representing a trade off that favors high utilization of the computing resource over the productivity of the people using the machines.

Below, we enumerate the key principles of DISC and how they differ from conventional supercomputer systems.

1. *Intrinsic, rather than extrinsic data.* Collecting and maintaining data are the duties of the system, rather than for the individual users. The system must retrieve new and updated information over networks and perform derived computations as background tasks. Users can use rich queries based on content and identity to access the data. Reliability mechanisms (e.g., replication, error correction) ensure data integrity and availability as part of the system function. By contrast, current supercomputer centers provide short-term, large-scale storage to their users, high-bandwidth communication to get data to and from the system, and plenty of computing power, but they provide no support for data management. Users must collect and maintain data on their own systems, ship them to the supercomputer for evaluation, and then return the results back for further analysis and updating of the data sets.

2. *High-level programming models for expressing computations over the data.* Current supercomputers must be programmed at a very low level to make maximal use of the resources. Wresting maximum performance from the machine requires hours of tedious optimization. More advanced algorithms, such as ones using sophisticated and irregular data structures, are avoided as being too difficult to implement even though they could greatly improve application performance. With DISC, the application developer is provided with powerful, high-level programming primitives that express natural forms of parallelism and that do not specify a particular machine configuration (e.g., the number of processing elements). It is then the job of the compiler and runtime system to map these computations onto the machine efficiently.

3. *Interactive access.* DISC system users are able to execute programs interactively and with widely varying computation and storage requirements. The system responds to user queries and simple computations on the stored data in less than one second, while more involved computations take longer but do not degrade performance for the queries and simple computations of others. By contrast, existing supercomputers are operated in batch mode to maximize processor utilization. Consequently, users generally maintain separate, smaller cluster systems to do their program development, where greater interaction is required. Inherently interactive tasks, such as data visualization, are not well supported. In order to support interactive computation on a DISC system, there must be some over-provisioning of resources. We believe that the consequent increased cost of computing resources can be justified based on the increased productivity of the system users.

4. *Scalable mechanisms to ensure high reliability and availability.* Current supercomputers provide reliability mainly by periodically checkpointing the state of a program, and then rolling back to the most recent checkpoint when an error occurs. More serious failures require bringing the machine down, running diagnostic tests, replacing failed components, and only then restarting the machine. This is an inefficient, nonscalable mechanism that is not suitable for interactive use. Instead, we believe a DISC system should employ nonstop reliability mechanisms, where all original and intermediate data are stored in redundant forms,

and selective recomputation can be performed in event of component or data failures. Furthermore, the machine should automatically diagnose and disable failed components; these would only be replaced when enough had accumulated to impair system performance. The machine should be available on a 24/7 basis, with hardware and software replacements and upgrades performed on the live system. This would be possible with a more distributed and less monolithic structure than is used with current supercomputers.

3 Comparison to Other Large-Scale Computer Systems

There are many different ways large-scale computer systems are organized, but DISC systems have a unique set of characteristics. We compare and contrast some of the other system types.

3.1 Current Supercomputers

Although current supercomputers have enabled fundamental scientific breakthroughs in many disciplines, such as chemistry, biology, physics, and astronomy, large portions of the scientific world really have not benefited from the supercomputers available today. For example, most computer scientists make use of relatively impoverished computational resources—ranging from their laptops to clusters of 8–16 processors. We believe that DISC systems could serve a much larger part of the research community, including many scientific areas, as well as other information-intensive disciplines, such as public health, political science, and economics. Moreover, even many of the disciplines that are currently served by supercomputers would benefit from the flexible usage model, the ease of programming, and the managed data aspects of DISC.

3.2 Transaction Processing Systems

Large data centers, such as those maintained by financial institutions, airlines, and online retailers, provide another model for large-scale computing systems. We will refer to these as *transaction processing systems*, since the term "data center" is too generic. Like a DISC system, the creation and maintenance of data plays a central role in how a transaction processing system is conceived and organized. High data availability is of paramount importance, and hence the data are often replicated at geographically distributed sites. Precise consistency requirements must be maintained, for example, so that only one person can withdraw the last dollar from a joint bank account. The creation and accessing of data at transaction processing systems occur largely in response to single transactions, each performing limited computation over a small fraction of the total data.

The strong set of reliability constraints placed on transaction systems [15], and the potentially severe consequences of incorrect operation (e.g., if the electronic banking system does not maintain consistent account balances), tend to narrow the range of implementation options and encourage fairly conservative design approaches. Large amounts of research have been done on increasing the amount of concurrency through distribution and replication of the data, but generally very complex and expensive systems are required to meet the combined goals of high capacity, availability, and reliability.

Search engines have very different characteristics from transaction processing systems that make it easier to get high performance at lower cost. A search engine need not track every change to every document on the web in real time. Users are satisfied if they have access to a reasonably large fraction of the available documents, and that the documents are reasonably up to date. Thus, search can be performed on cached copies of the web, with the updating of the caches performed as a background activity. In addition, the updating of the state is not done in response to individual transactions, but rather by an independent web crawling process. This decoupling between the agents that read data and those that update them makes it much easier to scale the size of the systems at a reasonable cost.

We envision that most DISC systems will have requirements and characteristics more similar to those of search engines than to transaction processing systems. Data sets for scientific applications typically have higher needs for accuracy and consistency than do search engines, but we anticipate they will not be updated as often or by as many agents as is the case for a transaction processing system. In addition, many DISC applications may be able to exploit relaxed consistency constraints similar to those found with web search. For those that extract statistical patterns from massive data sets, their outcomes will be relatively insensitive to small errors or inconsistencies in the data.

On the other hand, in contrast to transaction processing, many DISC operations will invoke large-scale computations over large amounts of data, and so they will require the ability to schedule and coordinate many processors working together on a single computation. All of these differences will lead to very different choices in DISC hardware, programming models, reliability mechanisms, and operating policies than is is found in transaction processing systems.

3.3 Grid Systems

Many research communities have embraced *grid computing* to enable a sharing of computational resources and to maintain shared data repositories [7]. Although the term "grid" means different things to different people, its primary form is a *computational grid* enabling a number of computers, often distributed geographically and organizationally, to work together on a single computational task. The low-bandwidth connectivity between machines typically limits this style of computation to problems that require little or no communication between the different subtasks. In addition, a *data grid* enables a shared data repository to be distributed across a number of machines, often in combination with a computational grid to operate on this data.

In many ways, our conception for DISC has similar objectives to a combined computational and data grid: we want to provide one or more research communities with an actively managed repository of shared data, along with computational resources that can create and operate on this data. The main difference between DISC systems and grid systems is physical: we believe that a DISC system benefits by locating substantial storage and computational power in a single facility, enabling much faster and much greater movement of data within the system. This makes it possible to support forms of computation that are completely impractical with grid systems. It also enables the system to be much more aggressive in using different forms of scheduling and load balancing to achieve interactive performance for many users, and to provide higher degrees of reliability and availability. We believe that we can provide more powerful programming models and better economies of scale by taking the more centralized approach to resource location and management represented by DISC.

Alex Szalay and Jim Gray stated in a commentary on 2020 Computing [25]:

> "In the future, working with large data sets will typically mean sending computations to the data, rather than copying the data to your workstation."

Their arguments were based on shear numbers: even the most optimistic predictions of network bandwidth do not allow transferring petabyte data sets from one site to another in a reasonable amount of time. The complexities of managing and computing over such data sets lend further impetus to the need to build systems centered around specific data repositories.

Over time, as multiple organizations set up DISC systems, we could well imagine creating a grid system with DISCs as the nodes. Through different forms of data mirroring and load sharing, we could mitigate the impact of major outages due to power failures, earthquakes, and other disasters.

4 Google: A DISC Case Study

We draw much of our inspiration for DISC from the infrastructure that companies have created to support web search. Many credit Inktomi (later acquired by Yahoo) for initiating the trend of constructing specialized, large-scale systems to support web search [9]. Their 300-processor system in 1998 pointed the way to the much larger systems used today. Google has become the most visible exemplar of this approach, and so we focus on their system as a case study. Their system demonstrates how a DISC system can be designed and utilized, although our view of DISC is much broader and envisions a more complex usage model than Google's. Our ideal system will almost certainly not match all of the characteristics of Google's.

Google has published a small, but high quality set of papers about their system design [6, 10, 13, 12, 14]. Although these papers set out a number of important concepts in system design, Google is fairly secretive about many specific details of their systems. Some of what is stated below is a bit speculative and may be wrong or out of date. Most likely, both Yahoo and Microsoft have comparable server infrastructure for supporting their search tools, but they are even more secretive than Google.

Google does not disclose the size of their server infrastructure, but reports range from 450,000 [21] to several million [20] processors, spread around at least 25 data centers worldwide. Machines are grouped into clusters of "a few thousand processors," with disk storage associated with each processor. The system makes use of low-cost, commodity parts to minimize per unit costs, including using processors that favor low power over maximum speed. Standard Ethernet communication links are used to connect the processors. This style of design stands in sharp contrast to the exotic technology found in existing supercomputers, using the fastest possible processors and specialized, high-performance interconnection networks, consuming large amounts of power and requiring costly cooling systems. Supercomputers have much higher interconnection bandwidth between their processors, which can dramatically improve performance on some applications, but this capability comes at a very high cost.

The Google system actively maintains cached copies of every document it can find on the Internet (around 20 billion), to make it possible to effectively search the entire Internet in response to each query. These copies must be updated on an ongoing basis by having the system *crawl* the web, looking for new or updated documents. In addition to the raw documents, the system constructs complex *index* structures, summarizing information about the documents in forms that enable rapid identification of the documents most relevant to a particular query. When a user submits a query, the front end servers direct the query to one of the clusters, where several hundred processors work together to determine the best matching documents based on the index structures. The system then retrieves the documents from their cached locations, creates brief summaries of the documents, orders them with the most relevant documents first, and determines which sponsored links should be placed on the page.

Processing a single query requires a total of around 10^{10} CPU cycles, not even counting the effort spent for web crawling and index creation. This would require around 10 seconds of computer time on a single machine (or more, when considering the time for disk accesses), but by using multiple processors simultaneously, Google generates a response in around 0.1 seconds [6].

It is interesting to reflect on how the Google server structure gets used, as indicated by our initial observation about the level of resources Google applies in response to often-mundane search queries. Some estimates say that Google earns an average of $0.05 in advertising revenue for every query to which it responds [17]. It is remarkable that they can maintain such a complex infrastructure and provide that level of service for such a low price. Surely we could make it a national priority to provide the scientific community with equally powerful computational capabilities over large data sets.

The Google hardware design is based on a philosophy of using components that emphasize low cost and low power over raw speed and reliability. They typically stay away from the highest speed parts, because these carry a price premium and consume greater power. In addition, whereas many processors designed for use in servers employ expensive hardware mechanisms to ensure reliability (e.g., the processors in IBM mainframes perform every computation on two separate data paths and compare the results), Google keeps the hardware as simple as possible. Only recently have they added error-correcting logic to their DRAMs. Instead, they make extensive use of redundancy and software-based reliability, following the lead set by Inktomi [9]. Multiple copies of all data are stored, and many computations are performed redundantly. The system continually runs diagnostic programs to identify and isolate faulty components. Periodically, these failed components are removed and replaced without turning the system off. (In the original server, the disk drives were held in with Velcro to facilitate easy replacement.) This software-based reliability makes it possible to provide different levels of reliability for different system functions. For example, there is little harm if the system occasionally fails to respond to a search query, but it must be meticulous about accounting for advertising revenue, and it must ensure high integrity of the index structures.

Google has significantly lower operating costs in terms of power consumption and human labor than do other data centers.

Although much of the software to support web crawling and search is written at a low level, they have implemented a programming abstraction, known as *MapReduce* [13], that supports powerful forms of computation performed in parallel over large amounts of data. The user needs only specify two functions: a *map* function that generates values and associated keys from each document, and a *reduction* function that describes how all the data matching each possible key should be combined. MapReduce can be used to compute statistics about documents, to create the index structures used by the search engine, and to implement their PageRank algorithm for quantifying the relative importance of different web documents. The runtime system implements MapReduce, handling details of scheduling, load balancing, and error recovery [12].

More recently, researchers at Google have devised programming support for distributed data structures they call *BigTable* [10]. Whereas MapReduce is purely a functional notation, generating new files from old ones, BigTable provides capabilities similar to those seen in database systems. Users can record data in tables that are then stored and managed by the system. BigTable does not provide the complete set of operations supported by relational databases, striking a balance between expressive power and the ability to scale for very large databases in a distributed environment.

In summary, we see that the Google infrastructure implements all the features we have enumerated for data-intensive super computing in a system tailored for web search. More recently, they have expanded their range of services to include email and online document creation. These applications have properties more similar to transaction processing than to web search. Google has been able to adapt its systems to support these functions successfully, although it purportedly has been very challenging for them.

5 Possible Usage Model

We envision that different research communities will emerge to use DISC systems, each organized around a particular shared data repository. For example, natural language researchers will join together to develop and maintain corpora from a number of different sources and in many different languages, plus derived statistical and structural models, as well as annotations relating the correspondences between phrases in different languages. Other communities might maintain finite-element meshes describing physical phenomena, copies of web documents, etc. These different communities will devise different policies for how data will be collected and maintained, what computations can be performed and how they will be expressed, and how different people will be given different forms of access to the data.

One useful perspective is to think of a DISC system as supporting a powerful form of database. Users can invoke operations on the database that can be simple queries or can require complex computations accessing large amounts of data. Some would be read-only, while others would create or update the stored data. As mentioned earlier, we anticipate that most applications of DISC will not have to provide the strong consistency guarantees found in the database support for transactions processing.

Unlike traditional databases, which support only limited forms of operations, the DISC operations could include user-specified functions in the style of Google's MapReduce programming framework. As with databases, different users will be given different authority over what operations can be performed and what modifications can be made.

6 Constructing a General-Purpose DISC System

Suppose we wanted to construct a general purpose DISC system that could be made available to the research community for solving data-intensive problems. Such a system could range from modest, say 1000 processors, to massive, say 50,000 processors or more. We have several models for how to build large-scale systems, including current supercomputers, transaction processing systems, and search-engine systems.

Assembling a system that can perform web search could build on standard hardware and a growing body of available software. The open source project *Hadoop* implements capabilities similar to the Google file system and support for MapReduce. Indeed, a near-term project to provide this capability as soon as possible would be worth embarking on, so that the university research community can become more familiar with DISC systems and their applications. Beyond web search, a system that performs web crawling and supports MapReduce would be useful for many applications in natural language processing and machine learning.

Scaling up to a larger and more general purpose machine would require a significant research effort, but we believe the computer science community would embrace such an effort as an exciting research opportunity. Below we list some of the issues to be addressed

- *Hardware Design.* There are a wide range of choices here, from assembling a system out of low-cost commodity parts, à la Google, to using off-the-shelf systems designed for data centers, to using supercomputer-class hardware, with more processing power, memory, and disk storage per processing node, and a much higher bandwidth interconnection network. These choices could greatly affect the system cost, with prices ranging between around \$2,000 to \$10,000 per node. In any case, the hardware building blocks are all available commercially. One fundamental research question is to understand the tradeoffs between the different hardware configurations and how well the system performs on different applications. Google has made a compelling case for sticking with low-end nodes for web search applications, but we need to consider other classes of applications as well. In addition, the Google approach requires much more complex system software to overcome the limited performance and reliability of the components. That might be fine for a company that hires computer science PhDs at the rate Google does, and for which saving a few dollars per node can save the company millions, but it might not be the most cost-effective solution for a smaller operation when personnel costs are considered.

- *Programming Model.* As Google has demonstrated with MapReduce and BigTable, there should be a small number of program abstractions that enable users to specify their desired computations at a high level, and then the runtime system should provide an efficient and reliable implementation, handling such issues as scheduling, load balancing, and error recovery. Some variations of MapReduce and BigTable would be good starts, but it is likely that multiple such abstractions will be required to support the full range of applications we propose for the system, and for supporting active collection and management of different forms of data.

One important software concept for scaling parallel computing beyond 100 or so processors is to incorporate error detection and recovery into the runtime system and to isolate programmers from both transient and permanent failures as much as possible. Historically, most work on and implementations of parallel programming assumes that the hardware operates without errors. By assuming instead that every computation or information retrieval step can fail to complete or can return incorrect answers, we can devise strategies to correct or recover from errors that allow the system to operate continuously. Work on providing fault tolerance in a manner invisible to the application programmer started in the context of grid-style computing [5], but only with the advent of MapReduce [13] and in recent work by Microsoft [19] has it become recognized as an important capability for parallel systems.

We believe it is important to avoid the tightly synchronized parallel programming notations used for current supercomputers. Supporting these forces the system to use resource management and error recovery mechanisms that would be hard to integrate with the interactive scheduling and flexible error handling schemes we envision. Instead, we want programming models that dynamically adapt to the available resources and that perform well in a more asynchronous execution environment. Parallel programs based on a task queue model [8] do a much better job of adapting to available resources, and they enable error recovery by re-execution of failed tasks. For example, Google's implementation of MapReduce partitions a computation into a number of map and reduce tasks that are then scheduled dynamically onto a number of "worker" processors. They cite as typical parameters having 200,000 map tasks, 4,000 reduce tasks, and 2,000 workers [12].

- *Resource Management.* A very significant set of issues concern how to manage the computing and storage resources of a DISC system. We want it to be available in an interactive mode and yet able to handle very large-scale computing tasks. In addition, even though it would be feasible to provide multiple petabytes of storage, some scientific applications, such as astronomy, could easily soak up all of this. Different approaches to scheduling processor and storage resources can be considered, with the optimal decisions depending on the programming models and reliability mechanisms to be supported.

  As described earlier, we anticipate multiple, distinct research communities to make use of DISC systems, each centered around a particular collection of data. Some aspects of the system hardware and support software will be common among these communities, while others will be more specialized.

- *Supporting Program Development.* Developing parallel programs is notoriously difficult, both in terms of correctness and to get good performance. Some of these challenges can be reduced by using and supporting high-level programming abstractions, but some issues, especially those affecting performance, affect how application programs should be written.

We must provide software development tools that allow correct programs to be written easily, while also enabling more detailed monitoring, analysis, and optimization of program performance. Most likely, DISC programs should be written to be "self-optimizing," adapting strategies and parameters according to the available processing, storage, and communications resources, and also depending on the rates and nature of failing components. Hopefully, much of this adaptation can be built into the underlying runtime support, but some assistance may be required from application programmers.

- *System Software.* Besides supporting application programs, system software is required for a variety of tasks, including fault diagnosis and isolation, system resource control, and data migration and replication. Many of these issues are being addressed by the Self-* systems project at Carnegie Mellon [2], but the detailed solutions will depend greatly on the specifics of the system organization.

Designing and implementing a DISC system requires careful consideration of a number of issues, and a collaboration between a number of disciplines within computer science and computer engineering. We are optimistic that we can form a team of researchers and arrive at a successful system design. After all, Google and its competitors provide an existence proof that DISC systems can be implemented using available technology.

Over the long term, there are many research topics that could be addressed by computer scientists and engineers concerning DISC systems. The set of issues listed previously will all require ongoing research efforts. Some additional topics include:

- *How should the processors be designed for use in cluster machines?* Existing microprocessors were designed to perform well as desktop machines. Some of the design choices, especially the exotic logic used to exploit instruction-level parallelism, may not make the best use of hardware and energy for systems that can make greater use of data parallelism. For example, a study by researchers at Google [6], indicated that their most critical computations did not perform well on existing microprocessors. Perhaps the chip area and power budgets would be better served by integrating many simpler processors cores on a single chip [4].

- *How can we effectively support different scientific communities in their data management and applications?* Clearly, DISC works for web search applications, but we need to explore how far and how well these ideas extend to other data-intensive disciplines.

- *Can we radically reduce the energy requirements for large-scale systems?* The power needs of current systems are so high that Google has set up a major new facility in Oregon, while Microsoft and Yahoo are building ones in Eastern Washington, to be located near inexpensive hydroelectric power [21]. Would a combination of better hardware design and better resource management enable us to reduce the required power by a factor of 10 or more?

- *How do we build large-scale computing systems with an appropriate balance of performance and cost?* The IT industry has demonstrated that they can build and operate very large and complex data centers, but these systems are very expensive to build (both machines and infrastructure) and operate (both personnel and energy). We need to create a framework by which system designers can rigorously evaluate different design alternatives in terms of their reliability, cost, and ability to support the desired forms of computation.

- *How can very large systems be constructed given the realities of component failures and repair times?* Measurements indicate that somewhere between 4% and 7% of the disks in a data center must be replaced each year [16, 24]. In a system with 50,000 disks, that means that disks will be failing every few hours. Even once a new unit is installed, it can take multiple hours to reconstruct its contents from the redundant copies on other disks, and so the system will always be involved in data recovery activities. Furthermore, we run the risk that all copies of some data item could be lost or corrupted due to multiple component failures. Creating reliable systems of such scale will require careful analysis of failure modes and frequencies, and devising a number of strategies for mitigating the effects of failures. We will require ways to assess the levels of criticality of different parts of the data sets in order to apply differential replication and recovery strategies.

- *Can we support a mix of computationally intensive jobs with ones requiring interactive response?* In describing our ideas to users of current supercomputers, this possibility has proved to be the one they find the most intriguing. It requires new ways of structuring and programming systems, and new ways to schedule their resources.

- *How do we control access to the system while enabling sharing?* Our system will provide a repository of data that is shared by many users. We cannot implement security by simply imposing complete isolation between users. In addition, we want more sophisticated forms of access control than simply whether a user can read or write some part of the data, since improper updating of the data could impede the efforts of other users sharing the data. We must guard against both accidental and malicious corruption.

- *Can we deal with bad or unavailable data in a systematic way?* When operating on very large data sets distributed over many disk drives, it is inevitable that some of the data will be corrupted or will be unavailable in a timely manner. Many applications can tolerate small amounts of data loss, and so they should simply skip over corrupted records, as is done in Google's implementation of MapReduce [13], and they should be allowed to proceed when enough data have been retrieved. Providing the right set of mechanisms to allow the application programmer to implement such strategies while maintaining acceptable accuracy requires ways to quantify acceptable data loss and clever design of the application-program interface.

- *Can high performance systems be built from heterogenous components?* Traditionally, most high-performance systems have been built using identical processors, disks, etc., in order to simplify issues of scheduling, control, and maintenance. Such homogeneity is required to support the tightly synchronized parallel programming models used in these systems, but would not be required for a more loosely coupled task queue model. Allowing heterogenous components would enable incremental upgrading of the system, adding or replacing a fraction of the processors or disks at a time.

Although the major search engine companies are examining many of these issues with their own systems, it is important that the university research community gets involved. First, there are many important disciplines beyond web search and related services that can benefit from the DISC principles. Second, academic researchers are uniquely equipped to bring these issues into a public forum where they can be systematically and critically evaluated by scholars from around the world. Companies are too driven by deadlines and too wary of protecting their proprietary advantages to serve this role.

In addition to being able to contribute to the progress of DISC, academics need to engage in this area to guarantee their future relevance. It is important that our students learn about the systems they will encounter in their careers, and that our research work addresses problems of real importance to the IT industry. There is a large and growing gap between the scale of systems found in academia compared to those of the numerous data centers worldwide supporting web search, electronic commerce, and business processes. Although some universities have large-scale systems in the form of supercomputers, only a small subset of academic computer scientists are involved in high performance computing, and these systems have very different characteristics from commercial data centers. Bringing DISC projects into university environments would provide new opportunities for research and education that would have direct relevance to the current and future IT industry.

7 Turning Ideas into Reality

We are convinced that DISC provides an important area for university-based research and education. It could easily spawn projects at multiple institutions and involve researchers in computer engineering, computer science, computational science, and other disciplines that could benefit from the availability of DISC systems.

How then should we proceed? There are many possible paths to follow, involving efforts of widely varying scale and scope. Choosing among these will depend on the availability of research funding, how many institutions will be involved, and how collaborative their efforts will be. Rather than pin down a specific plan, we simply describe possible options here.

One factor is certain in our planning—there are many researchers who are eager to get involved. I have spoken with researchers in a number of companies and universities, and there is a clear consensus that there are ample opportunities for exciting work ranging across the entire spectrum of computing research disciplines.

7.1 Developing a Prototype System

One approach is to start by constructing a prototype system of around 1000 processing nodes. Such a system would be large enough to demonstrate the performance potential of DISC and to encounter some of the challenges in resource management and error handling. For example, if we provision each node with at least one terabyte of storage (terabyte disks will be available within the next year or so), the system would have a storage capacity of over one petabyte. This would easily provide enough storage to hold replicated copies of every document available over the worldwide web. By having two dual-core processors in each node, the resulting machine would have 4,000 total processor cores.

In order to support both system and application researchers simultaneously, we propose constructing a system that can be divided into multiple partitions, where the different partitions could operate independently without any physical reconfiguration of the machines or interconnections.

Typically, we would operate two types of partitions: some for application development, focusing on gaining experience with the different programming techniques, and others for systems research, studying fundamental issues in system design. This multi-partition strategy would resolve the age-old dilemma of how to get systems and applications researchers working together on a project. Application developers want a stable and reliable machine, but systems researchers keep changing things.

For the program development partitions, we would initially use available software, such as the open source code from the Hadoop project, to implement the file system and support for application programming.

For the systems research partitions, we would create our own design, studying the different layers of hardware and system software required to get high performance and reliability. As mentioned earlier, there is a range of choices in the processor, storage, and interconnection network design that greatly affects the system cost. During the prototyping phases of the project, we propose using relatively high-end hardware—we can easily throttle back component performance to study the capabilities of lesser hardware, but it is hard to conduct experiments in the reverse direction. As we gain more experience and understanding of tradeoffs between hardware performance and cost, and as we develop better system software for load balancing and error handling, we may find that we can build systems using lower-cost components.

Over time, we would migrate the software being developed as part of the systems research to the partitions supporting applications programming. In pursuing this evolutionary approach, we must decide what forms of compatibility we would seek to maintain. Our current thinking is that any compatibility should only be provided at a very high level, such that an application written in terms of MapReduce and other high-level constructs can continue to operate with minimal modifications, but complete compatibility is not guaranteed. Otherwise, our systems researchers would be overly constrained to follow nearly the exact same paths set by existing projects.

We can estimate the hardware cost of a prototype machine based on per-node costs. Our current thinking is that it would be best to use powerful nodes, each consisting of a high-end rack-mounted server, with one or two multicore processors, several terabytes of disk, and one or more high-performance communications interfaces. Going toward the high end would create a more general prototyping facility. We estimate such nodes, plus the cost of the high performance communication network, would cost around \$10,000, yielding a hardware cost of around \$10 million for the machine.

In addition to the cost of procuring hardware, we would incur costs for personnel, infrastructure, and energy. Since one goal is to develop and maintain software of sufficiently high quality to provide a reliable computing platform, we must plan for a large staff (i.e., not just graduate students and faculty), including software architects, programmers, project managers, and system operators. The exact costs depend to a large extent on issues such as where the work is performed, where the system is located, and how many researchers get involved.

7.2 Jump Starting

Instead of waiting until hardware can be funded, procured, and installed, we could begin application development by renting much of the required computing infrastructure. Recently, Amazon has begun marketing network-accessible storage, via its Simple Storage System (S3) service, and computing cycles via its Elastic Computing Cloud (EC2) service [11]. The current pricing for storage is \$0.15 per gigabyte per day (\$1,000 per terabyte per year), with addition costs for reading or writing the data. Computing cycles cost \$0.10 per CPU hour (\$877 per year) on a virtual Linux machine. As an example, it would be possible to have 100 processors running continuously, maintaining a 50 TB data set, updated at a rate of 1 TB per day at a cost of \$214,185 per year. That, for example, would be enough to collect, maintain, and perform computations over a substantial fraction of the available web documents.

We view this rental approach as a stopgap measure, not as a long-term solution. For one thing, the performance of such a configuration is likely to be much less than what could be achieved by a dedicated facility. There is no way to ensure that the S3 data and the EC2 processors will be in close enough proximity to provide high speed access. In addition, we would lose the opportunity to design, evaluate, and refine our own system. Nevertheless, the view this capability as an intriguing direction for computing services and a way to move forward quickly.

7.3 Scaling Up

An important goal in building a 1000-node prototype would be to determine the feasibility and study the issues in constructing a much larger system, say 10,000 to 50,000 nodes. Scaling up to such a large system only makes sense if we can clearly demonstrate the ability of such a system to solve problems having high societal importance, and to do so more effectively than would be possible with other approaches. We would also need to understand whether it is best to create a small number of very large machines, a larger number of more modest machines, or some combination of the two. Having the 1000-node prototype would enable us to study these issues and make projections on the scalability and performance of our designs.

8 Conclusion

As we have described the ideas behind DISC to other researchers, we have found great enthusiasm among both potential system users and system developers. We believe it is time for the computer science community to step up their level of thinking about the power of data-intensive computing and the scientific advances it can produce. Just as web search has become an essential tool in the lives of people ranging from schoolchildren to academic researchers to senior citizens, we believe that DISC systems could change the face of scientific research worldwide.

We are also confident that any work in this area would have great impact on the many industries that benefit from more powerful and more capable information technology. In domains ranging from retail services to health care delivery, vast amounts of data are being collected and analyzed. Information can be extracted from these data that makes companies better serve their customers while running more efficiently and that detects long term health trends in different populations. The combination of sensors and networks to collect data, inexpensive disks to store data, and the benefits derived by analyzing data causes our society to be increasingly data intensive. DISC will help realize the potential all these data provides.

Universities cannot come close to matching the capital investments that industry is willing to undertake. In 2006, Microsoft announced plans to invest $2 billion in server infrastructure for 2007 [21], and Google $1.5 billion. It should be realized, though, that these companies are trying to serve the needs of millions of users, while we are only supporting hundreds or thousands. In addition, the federal government spends billions of dollars per year for high-performance computing. Over the long term, we have the opportunity to help that money be invested in systems that better serve the needs of their users and our society. Thus, getting involved in DISC is within the budgetary reach of academic computer scientists.

In this research area, universities are in the unusual position of following a lead set by industry, rather than the more normal reverse situation. Google and its competitors have demonstrated a new style of computing, and it is important for universities to adopt and build on these ideas. We have the ability to develop and evaluate ideas systematically and without proprietary constraints. We can apply these ideas to domains that are unlikely to produce any commercial value in the near term, while also generating technology that has long-term economic impact. We also have a duty to train students to be at the forefront of computer technology, a task we can only do by first moving to the frontier of computer systems ourselves.

References

[1] Google tops translation ranking. *News@Nature*, Nov. 6, 2006.

[2] M. Abd-El-Malek, W. V. Courtright II, C. Cranor, G. R. Ganger, J. Hendricks, A. J. Klosterman, M. Mesnier, M. Prasad, B. Salmon, R. R. Sambasivan, S. Sinnamohideen, J. D. Strunk, E. Thereska, M. Wachs, and J. J. Wylie. Early experiences on the journey towards self-* storage. *IEEE Data Eng. Bulletin*, 29(3):55–62, 2006.

[3] V. Akcelik, J. Bielak, G. Biros, I. Epanomeritakis, A. Fernandez, O. Ghattas, E. J. Kim, J. Lopez, D. R. O'Hallaron, T. Tu, and J. Urbanic. High resolution forward and inverse earthquake modeling on terasacale computers. In *Proceedings of SC2003*, November 2003.

[4] K. Asanovic, R. Bodik, B. C. Catanzo, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, U. C., Berkeley, December 18 2006.

[5] Ö. Babaŏglu, L. Alvisi, A. Amoroso, R. Davoli, and L. A. Giachini. Paralex: an environment for parallel programming in distributed systems. In *6th ACM International Conference on Supercomputing*, pages 178–187, 1992.

[6] L. A. Barroso, J. Dean, and U. Hölze. Web search for a planet: The Google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

[7] F. Berman, G. Fox, and T. Hey. The Grid: Past, present, and future. In F. Berman, G. C. Fix, and A. J. G. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, pages 9–50. Wiley, 2003.

[8] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. Cilk: An efficient multithreaded runtime system. *ACM SIGPLAN Notices*, 30(8):207–216, August 1995.

[9] E. A. Brewer. Delivering high availability for Inktomi search engines. In L. M. Haas and A. Tiwary, editors, *ACM SIGMOD International Conference on Management of Data*, page 538. ACM, 1998.

[10] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. BigTable: A distributed storage system for structured data. In *Operating Systems Design and Implementation*, 2006.

[11] T. Claburn. In Web 2.0 keynote, Jeff Bezos touts Amazon's on-demand services. *Information Week*, Apr. 17, 2007.

[12] J. Dean. Experiences with MapReduce, an abstraction for large-scale computation. In *International Conference on Parallel Architecture and Compilation Techniques*. ACM, 2006.

[13] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Operating Systems Design and Implementation*, 2004.

[14] S. Ghemawat, H. Gobioff, and S. T. Leung. The Google file system. In *Symposium on Operating Systems Principles*, pages 29–43. ACM, 2003.

[15] J. Gray. The transaction concept: Virtues and limitations. In *Very Large Database Conference*, pages 144–154, 1981.

[16] J. Gray and C. van Ingen. Empirical measurements of disk failure rates and error rates. Technical Report MSR-TR-2005-166, Microsoft Research, 2005.

[17] M. Helft. A long-delayed ad system has Yahoo crossing its fingers. *New York Times*, Feb. 5, 2007.

[18] T. Hey and A. Trefethen. The data deluge: an e-Science perspective. In F. Berman, G. C. Fix, and A. J. G. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, pages 809–824. Wiley, 2003.

[19] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *EuroSys 2007*, March 2007.

[20] J. Markoff. Sun and IBM offer new class of high-end servers. *New York Times*, Apr. 26, 2007.

[21] J. Markoff and S. Hansell. Hiding in plain sight, Google seeks more power. *New York Times*, June 14, 2006.

[22] T. M. Mitchell, R. Hutchinson, R. S. Niculescu, F. Pereira, X. Wang, M. Just, and S. Newman. Learning to decode cognitive states from brain images. *Machine Learning*, 57(1–2):145–175, October 2004.

[23] L. Ren, G. Shakhnarovich, J. K. Hodgins, H. Pfister, and P. Viola. Learning silhouette features for control of human motion. *ACM Transactions on Graphics*, 24(4), October 2005.

[24] B. Schroeder and G. A. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *FAST'07: Fifth USENIX Conference on File and Storage Technlogies*, 2007.

[25] A. Szalay and J. Gray. Science in an exponential world. *Nature*, 440, March 23 2006.

[26] A. S. Szalay, P. Z. Kunszt, A. Thakar, J. Gray, D. Slutz, and R. J. Brunner. Designing and mining multi-terabyte astronomy archives: The Sloan Digital Sky Survey. In *SIGMOD International Conference on Management of Data*, pages 451–462. ACM, 2000.

A Acknowledgments

A number of people have contributed their ideas and helped refine my ideas about DISC. The following is a partial list of people who have been especially helpful.

- Carnegie Mellon University
  - Guy Blelloch, for insights on parallel programming
  - Jamie Callan and Jaime Carbonell, for explaining the requirements and opportunities for language translation and information retrieval.

  - Greg Ganger and Garth Gibson, for all aspects of DISC system design and operation
  - Peter Lee, Todd Mowry, and Jeannette Wing, for high-level guidance and support
  - Tom Mitchell, for insights into the needs and capabilities of machine learning
  - David O'Hallaron, for devising the domain-specific database perspective of DISC operation
  - Anthony Tomasic, for information on current commercial data centers.

- Other universities
  - David Patterson (Berkeley), Ed Lazowska (Washington) for insights into the role of university research in this area.
  - Tom Andersen (Washington), regarding how this project relates to the proposed GENI project.

- Google
  - Urs Hölzle, for convincing us that it would be feasible for universities to build systems capable of supporting web crawling and search.
  - Andrew Moore, for advice on the resources required to create and operate a DISC system

- Intel
  - George Cox, for describing how important cluster computing has become to Intel.

- Microsoft
  - Sailesh Chutani, for ideas on some major challenges facing large-scale system design, such as the high power requirements.
  - Tony Hey, for a perspective on scientific computing
  - Roy Levin, for a historical perspective and for calling for a more precise characterization of the nature of problems suitable for DISC.
  - Rick Rashid, for a "reality check" on the challenges of creating and maintaining large-scale computing facilities

- Sun
  - Jim Waldo for encouraging us to think about heterogenous systems.

- Elsewhere
  - Bwolen Yang, for an appreciation of the need to create systems that are resilient to failures.
  - Tom Jordan (Southern California Earthquake Center), for insights on operating a multi-institutional, data and computation-intensive project.