Research

My research has focussed on large-scale systems, including an application-specific framework for geographic environmental modeling systems (GEMS), network-attached secure disks (NASD), and Active Disks, storage devices with the capability to run application-specific code. The fundamental issues spanning these projects were how to 1) partition computation among heterogeneous nodes in the system, 2) scale the system to a large numbers nodes, and 3) operate on large volumes of data efficiently. The core problem that ties these projects together the specification, and more importantly the analysis, required as part of the "functional model" of large systems. What mechanisms can we provide to describe the behavior of the components of a system that allows enough application-specific "customization" while introducing enough constraints to provide some hope of understanding overall system behavior?

Active Disks, intelligent storage, parallel programming, databases

In my thesis work, I propose Active Disks, individual disk drives that provide an environment for executing application-level code. This recognizes that at some point a "general-purpose" interface is no longer sufficient and the most efficient way to manage a system interface is to allow flexibility on both sides. A modern disk drive is simply a small computer. A current high-end disk drives contains a Motorola 68020 processor at 40 MHz, 4 MB of RAM, and a 40 MB/s network, along with some spinning magnetic material for permanent storage. Trends in the design of drive control chips promise processors at upwards of 200 MHz [Siemens, product announcement] and 16 MB of RAM [Seagate, workshop presentation]. In a large SMP database server with 50 disks, this means the drives will have an aggregate computing power up to 5x to 10x what is available in the host processors. And, even more importantly, the I/O backplane of such a system cannot deliver anywhere near the aggregate bandwidth of fifty 20 MB/s disks [Riedel98, Usenix NT]. This means that application-specific processing that can take advantage of this parallel computing power and reduce the amount of data traffic on the network, can significantly improve performance and scalability [Riedel97, technical report]. My thesis contributes to this area in three major ways: by demonstrating a set of important data-intensive applications that benefit from such an architecture [Riedel98, VLDB]; providing a model for evaluating the performance of running on Active Disks compared to a traditional system; and describing the characteristics and limitations of applications that are candidates for Active Disks. My work focussed on specific applications from database, data mining, and multimedia. I have implemented these in a prototype system showing 2x performance gains for a small system, with scalability to 10x and 20x for more typical database servers.

Storage systems, networking, file systems, distributed systems

In the Network-Attached Secure Disks (NASD) project, we suggest that individual disk drives be made first-class citizens on the general-purpose network, rather than being "stuck" behind file servers on a special-purpose peripheral "network". Clients are able to transfer data directly from drives, and the full network bandwidth is available for these transfers [Gibson98, ASPLOS]. File servers become simply "file managers" and are responsible only for policy decisions and metadata management [Gibson97, SIGMETRICS]. We also propose modifying the interface to storage to support autonomous devices by providing storage "objects" that drives can manage with additional semantics [Gibson97, technical report]. This improves the flexibility available to "intelligence" at the devices and off-loads the server. As part of this work, we also studied workloads in distributed file systems [Riedel96, MSS&T], scientific applications, and worked with the Scalable I/O Initiative [Corbett96, Supercomputing]. Our work on network-attached disks has contributed to the current industry push for "network storage" and our proposal for "object-oriented disks" is the basis for a recent proposal to the SCSI standards committee.

Application-specific frameworks, object-oriented systems

While working on my Master's degree, I was supported as part of a Grand Challenge grant in Environmental Modeling. The goal of the group that I led was to develop an application-specific framework for geographic environmental modeling systems (GEMS) that could later be applied to similar application areas [Riedel94, ECOOP]. We chose to pursue an object-oriented design methodology and the final project combined aspects of user interface, visualization, parallel computing, and data management. This experience provided me with my first insight into the process of building a large computer system [Bruegge95, IEEE] as we dealt with issues of user requirements (often ill-specified, or unknown), performance, and the interaction among heterogeneous systems (for display, computation, and storage).

Challenges & Future Work

The biggest challenge to the wide applicability of Active Disks is the way code is written and distributed across hosts and disks. The use of standard interfaces such as SCSI or NASD provides flexibility for optimization above and beneath the interface, but allows information to flow across the interface only in very specific ways. On the other hand, providing general programmability at the device allows infinite customization of the interface, but also allows infinite ability to make bad partitioning decisions. Object-oriented design breaks a system into the object model, the dynamic model, and the functional model. When I last taught object-oriented design (admittedly over three years ago) the methodologies were very good at the object model, and this was very powerful - class hierarchies, inheritance, and specialization help to decompose a problem and tie it to "real world" objects. The dynamic model was mainly used for user interface design. The functional model was often neglected, and this is exactly where the core behavior of the system was to be specified. How do a particular set of classes interact with each other? What is the communication between decomposed modules? What are the performance requirements? All of these things were part of the functional model as I understood it. This was where the least amount of knowledge (in terms of techniques we could teach or learn from) was, and where the least amount of effort on a project was expended. It is also where the hardest part of the system design lies. How can we describe the behavior of a system in a way that allows us to control, or at least analyze, performance in general, as well as the "ilities" - scalability, reliability, manageability?

My future research work will focus on this aspect of systems development, working from the context of active storage systems. A server that manages a large number of Active Disks ties together the problems of distributed computing, programming languages, parallel computing, and fault tolerance (disks must continue to preserve the integrity of user data). Starting from storage systems provides a key point of leverage as current interfaces to storage are relatively limited and well-defined, and the question becomes how much or how fast to pull things "across" these interfaces. As more and more data becomes computerized and widely available, storage will also become increasing important - in large-scale databases, in data mining, in Web systems, and in multimedia. Critical issues to address include: How can we aid programmers in re-partitioning their code for Active Disks? Are there general "design patterns" that can be codified? Are there general performance models that can be applied before a system is built (rather than requiring profiling or extensive "tuning" after the fact)? Can we define a set of measurable core properties that a distributed component must have in order to perform well as part of a whole (communication/compute ratio limited to X, or variance in latency limited to Y)? Or particular properties that it should not have (single point of failure, memory requirement of Z)? Can this be done across all the important "performance" characteristics - including basic "throughput", but also reliability and manageability?

Teaching

In addition to this research work, I have been a teaching assistant four times for three different faculty members. Three of these were project-based courses in Software Engineering and Advanced Software Engineering that teach the basics of software engineering techniques and object-oriented design and then "sets the students loose" on a large development project that they complete as a team. The project brings in a "client" from outside the university who has a particular software problem they want solved - when I taught these included a mobile system for emergency management first responders, a wireless healthcare system, and an aircraft reservation system. This class exposes students to working in teams and dealing with the issues of having a real client, real system issues, and real deadlines. Many students have contacted us after taking this course and told us that this was the single best experience that helped them during job interviews and in their eventual software development positions. I also taught a senior-level undergraduate and first-year graduate course in systems architecture that started from the premise "you've learned all about how processors work, now let's talk about the rest of the system." We taught the details of the memory hierarchy, basic performance evaluation, performance tuning, and touched on storage systems and supercomputer architectures. I believe the most successful parts of these courses was the "hands on" approach and the direct relationships we drew between the course work and the issues the students would face in their jobs, and I hope to continue that in my own teaching. One of the best-received topics in the systems architecture class was the storage lecture that I gave and the lecture on fault tolerance given by another guest speaker. Storage and I/O issues are becoming more and more important in real systems and in industry focus (as we reach the limits of what can be gained from faster CPUs, data bases continue to grow, and the "connectedness" of data continues to increase). I believe these topics need to form a larger and more important part of both computer science and computer engineering curricula in the future, and this is something I will work to promote.