Research Statement

Rajesh Krishna Balan
rajesh@cs.cmu.edu
http://www.cs.cmu.edu/~rajesh/research/

My interests lie at the intersection of mobile computing, operating systems, software engineering, and usability testing. My thesis research has concentrated on the problem of allowing large computationally intensive applications to run on resource limited mobile devices such as cellphones, PDAs, and wrist watches. In addition to this work, I have also developed a distributed infrastructure to support massively multiplayer online games and designed various extensions to TCP that make it either more energy efficient or work better in wireless environments.

1 Research Philosophy

My research philosophy revolves around the belief that successful systems research has to demonstrate two things: 1) a novel and interesting approach to an important research problem, and 2) usability. I strongly believe that for a system to be successful, it has to be usable by the target audience. On the other hand, a highly usable system that doesn't advance the state of the art in any way is not particularly interesting from a research perspective either.

My research style is strongly systems-oriented with a large emphasis on hands-on system implementation. A real implementation is crucial, in my opinion, for testing complicated distributed systems as the complexity inherent in these systems is incredibly hard to capture by simulation studies alone. A real implementation is also vital in validating the usability of the system. In addition, the software artifacts from such implementations are important contributions that drive future research and even product development in the area. I am also a strong proponent of collaborative research. Modern systems are so complicated that it is impossible for one person to have all the answers to every problem encountered. As such, I tend to work with about 2-4 other people, each with slightly different skills and backgrounds, to tackle a large systems problem.

This philosophy and style is evident in all the research projects I have been involved with. For example, my thesis uses a novel insight, called *tactics*, to build a powerful remote execution system, called *Chroma*, for mobile devices. Chroma uses remote execution to allow large applications to run on limited mobile devices and I conducted numerous experiments to validate its performance. In addition, I also developed a software engineering process, called *RapidRe*, that allows software developers to rapidly retarget their applications to use Chroma. To validate RapidRe, I conducted, using Human Computer Interaction techniques, a rigorous software usability study.

I believe that my skill set of a rigorous systems background coupled with software engineering and detailed usability testing experience leave me uniquely placed to develop novel high performance software systems that are provably usable by developers and end users. The next few sections describe some of my recent work.

2 A Dynamic Adaptive Runtime System for Mobile Computing

A key challenge in mobile computing is to enable resource limited mobile devices to run large computationally intensive, yet highly useful, applications such as language translators and speech recognizers. One solution is to use remote execution to augment the resource capabilities of these mobile devices. For my thesis, I addressed the following two key research questions involved in building a remote execution system for mobile devices:

- What are the viable ways to partition an application for remote execution?
- At runtime, how should we decide which of the viable partitions should be selected?

The first problem was to determine the viable partitions of an application. In general, an application can be partitioned in arbitrarily many ways. Clearly, arbitrary partitioning is intractable in practice for a dynamic runtime system. However, a careful study of a large class of useful mobile applications revealed that the number of useful application partitions is actually very small. These partitions are useful because a) they partition the computationally intensive portions of the application, and b) they are large enough to amortize the partitioning overheads. I call these useful partitions the *tactics* of an application. These tactics differ in the amount of resources they use and in application quality. For example, tactic A may require less memory than tactic B but generate a less accurate result.

However, how can an application's tactics be identified? Do we need to examine large amounts of application code to identify these small number of tactics? Fortunately, this is not the case. My experiments have shown that application developers can specify the tactics for large unfamiliar applications in a very short amount of time. Tactics make dynamically deciding the appropriate application partitioning a tractable proposition. To utilize this capability, I built an adaptive runtime system for mobile devices, called *Chroma*, that dynamically partitions large applications.

At the heart of Chroma is a smart solver that decides at runtime which application tactic to use. The solver uses three inputs (in addition to the list of tactics) to make its decision. First, it uses a number of resource measurers to obtain the current resource availability in the environment. For example, the number of available remote servers and the available memory and CPU cycles on each of these servers. Next, the solver predicts the resource usage of the application using Narayanan's dynamic resource prediction techniques [8]. Finally, the solver uses utility functions (mathematical expressions of resource constraints) to pick the optimal tactic that a) does not exceed the available resources, and b) satisfies the user's preferences (such as high accuracy results or low battery consumption). To obtain accurate utility functions, I integrated Chroma with a system, called *Prism* [11], that monitors the user and generates appropriate utility functions that match the user's preferences. A paper (currently available as a technical report [12]) describing, in detail, this integration is being prepared for submission to a top Software Engineering journal.

An unexpected benefit of tactics is that it allows Chroma to automatically benefit from resource rich environments. For example, the mobile user enters a smart room containing a large number of idle servers. In such environments, Chroma is able to use extra servers to obtain higher application quality, reduce the application's latency, or both. Chroma was first presented at the SIGOPS European Workshop 2002 [2]. A more detailed paper, containing extensive experimental validation showing that Chroma can achieve excellent application performance in mobile environments, was presented at MobiSys 2003 [5]. A masters students also extended the MobiSys validation and focused solely on the benefits of utilizing extra server resources. These results are available as a technical report [6].

3 Enabling Rapid Retargeting of Existing Applications

Unfortunately, developing powerful mechanisms to support dynamic remote execution is only a partial solution. An equally important problem is reducing the effort needed to modify existing applications to use these mechanisms. Since mobile devices have very short "on-the-market" lifespans (of about 6 months to 1 year on average), it is important that existing applications can be rapidly retargeted for these devices. Prior experience with systems such as Rover and Odyssey suggests that it takes about two to four weeks to retarget an existing application to use these adaptive systems. These times are much too large and make any large scale application support for these systems extremely tedious.

Solving this problem is difficult as existing applications are written in a variety of programming languages including C, C++, Java, and even Ada and Matlab. Hence, it is not possible to reuse existing code analysis techniques to develop a fully automatic retargeting solution as these techniques do not support all these languages.

The key insights to my solution were the recognition that a) it is possible to cleanly separate the application-specific knowledge needed to partition the application from the general mechanisms (such as resource measurers etc.) needed to choose and execute the optimal runtime partition, and b) that this application-specific knowledge can be expressed in a simple compact declarative form.

Using these insights, I developed a retargeting process, called *RapidRe*, that allows developers to rapidly retarget their applications to support dynamic remote execution. RapidRe is language independent and ensures that once an application is retargeted, it does not need to be retargeted again if the hardware or underlying runtime system changes.

RapidRe consists of a) a domain specific language, called *Vivendi*, b) the Chroma runtime (that provides the general mechanisms needed for adaptation), and c) a smart stub generation tool. Developers specify the application-specific information (such as the tactics) using Vivendi. This description is then processed by the stub generator which creates most of the code needed to interface the application with Chroma. The stub also generates application-specific APIs that need to be used by the developer to create the final client and server components of the retargeted application.

To validate RapidRe, I conducted a rigorous software usability study using 13 undergrads and 8 applications (written in a variety of languages including C, C++, Java, Ada, and Tcl/Tk). The applications were all real applications, either downloaded from the Internet or developed by other research groups, that were useful for mobile users. They included a speech recognizer, a language translator, an augmented reality application, an optical character recognizer, and a face recognizer. The results of the usability study were very impressive. They showed that even novice developers could successfully retarget large complex applications in under 4 hours. In addition, the quality of the retargeted applications was not sacrificed either. Most of the retargeted applications performed just as well as applications hand-retargeted by an expert developer. This work has been initially published as a technical report [3]. We are currently preparing it for submission to a top Software Engineering venue. A more detailed paper, describing both Chroma and RapidRe, is being prepared for submission to the ACM TOCS journal.

4 Supporting Massively Multiplayer Online Games

During my internship at IBM Research Watson, I worked with Maria Ebling, Paul Castro, and Archan Misra (all from IBM Research Watson) to develop a distributed middleware system, called Matrix, for massively multiplayer games (MMG). This is a very demanding problem as MMGs can have up to one million players, distributed among up to 10,000 servers, participating in a shared virtual environment.

Our challenge was to develop a distributed infrastructure that could support such a huge virtual environment. This infrastructure would distribute the virtual environment across a large number of servers. We concentrated on three particular problems associated with doing this; 1) mechanisms to automatically ensure that all servers in Matrix were as consistent with each other as they needed to be, 2) mechanisms to automatically handle excess load, caused by hot spots or other factors, on any server, and 3) ensure that it is easy and attractive for game developers to use Matrix.

To solve the consistency problem, we used the insight that multiplayer games are nearly decomposable to develop the concept of overlap regions. These overlap regions clearly define which servers need to be informed of events occurring in any part of the virtual environment. Hence, this stops Matrix from unnecessarily sending packets to servers that do not need to see those packets.

To solve the load problem, we developed a mechanism that allows servers to be automatically added and removed from Matrix. With this mechanism, if an existing server starts experiencing an unacceptably high load, Matrix will automatically add a new server to share the load. This process continues until the load on all servers is below a specified threshold. We developed the appropriate mechanisms to move part of the virtual world (and the corresponding game clients) quickly and effectively to the new servers. Conversely, if the game becomes underutilized, Matrix can reclaim underutilized servers and combine portions of the virtual world onto other servers.

Finally, we ensured that Matrix was attractive to game developers by a) retaining the existing client-server communication model favored by game developers, and b) by providing a set of simple APIs that allow game developers to easily modify their games to use Matrix.

To validate our solutions, I wrote, in C, a complete implementation of Matrix. I then downloaded three real multiplayer games and modified them, in under 16 hours each, to use Matrix. These games were Bzflag [9], a tank shooting game, Daimonin [13], a role playing game, and Quake 2 [7], a shooting game. I then conducted numerous experiments, including small scale deployment measurements, microbenchmarks, and a user study (together with Maria), that showed that Matrix a) performed well under even extreme load conditions, b) had reasonable overheads, and c) did not introduce any human-visible latency. Archan and I also performed a simple analytical analysis that showed that the I/O bandwidth of individual servers was the limiting factor in Matrix's ability to scale to one million users. This work was presented as a short paper at Middleware 2005 [1]. A more complete description of Matrix, containing all the evaluation results, was published as an IBM technical report [4] and also submitted to the IEEE TC journal.

5 Future Interests

My prior research has raised a number of potential areas for future research in the broad area of mobile and distributed computing. In the short term, I hope to deploy Chroma on real mobile devices. This deployment will test if Chroma can indeed support the unexpected behavior and frequently changing requirements of a real user. I also plan to extend RapidRe to target the usability of other adaptive systems such as peer-to-peer systems, web services, self-configuring systems, and adaptive systems for grid computing by developing a range of domain-specific languages (similar to Vivendi) and stub generators. This would allow developers to quickly retarget useful applications, that match certain characteristics, for use in these other domains.

In the longer term, I plan to direct my research focus towards the pursuit of high performing usable systems for emerging domains. For example, even though the rapid proliferation of mobile devices has made these once rare and expensive devices a common and even integral part of our everyday lives, these devices are still massively underutilized. In particular, they still cannot use resources in the environment to augment their hardware limitations. This would allow them to execute many more user-desired applications, such as language translators and speech recognizers, with high quality and low latency. One reason is that it is currently not possible to securely and confidently use random servers in the environment. For example, how do you securely start application servers on machines you do not control and how do you detect malicious servers? I believe that virtual machine technology and secure hardware support (such as Trusted Platform Modules [10]) provide us with the basic building blocks for a possible solution.

Another deficiency is that managing adaptive runtime systems on mobile devices currently requires too much user attention. Users constantly have to micromanage the system to tell it which resources to prefer and what application settings to use. Eventually, the user will stop using the device entirely. I believe that the solution is to develop system level support for unobtrusive yet highly accurate user preference monitoring that can be used to automatically drive these adaptive systems (such as Chroma). This, in turn, will allow mobile devices to run applications without constantly interrupting the user (except for the rare cases where the monitoring system fails). Successfully building this support will require answering a number of questions. For example, are user actions in previously encountered mobile situations mostly predictable? If so, is a combination of history and active monitoring of user actions (which application is being run, which files etc.) sufficient? If not, what system support do we need to account for this unpredictable behavior? Ultimately, the answers to these questions will have to be determined through user deployments and testing.

Finally, I plan to continue my initial work in building infrastructure for MMGs because a) the number of MMG players is expected to grow rapidly in the next few years [14], and b) the current deployed solutions require a large amount of maintenance and don't scale well. My goal is to build a reliable, low maintenance, distributed infrastructure that can support up to 1,000,000 concurrent game players without adding any game player perceived latency. In addition, based on my experience with Matrix, the infrastructure also has to be usable (it should be easy to modify existing games to use the infrastructure) and secure (the infrastructure should not make it easier for players to cheat) before game developers will use it. I believe that this research thrust will result in many insights in consistency update and failure recovery mechanisms, load balancing and packet filtering technology, and in tools for rapid retargeting.

References

[1] R. K. Balan, M. Ebling, P. Castro, and A. Misra. Matrix: Adaptive middleware for distributed multiplayer games. In *Proceedings of the* 6^{th} *ACM/IFIP/USENIX International Middleware Conference (Middleware)*, Nov. 2005. Short Paper.

[2] R. K. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang. The case for cyber foraging. In *Proceedings of the* 10^{th} *ACM SIGOPS European Workshop*, pages 87–92, Saint Emillion, France, Sept. 2002.

[3] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb. Simplifying cyber foraging for mobile devices. Technical Report CMU-CS-05-157, Carnegie Mellon University, Pittsburgh, PA, Aug. 2005.

[4] R. K. Balan, A. Misra, M. Ebling, and P. Castro. Matrix: Adaptive middleware for distributed multiplayer games. Technical Report RC23764, IBM Research Watson, Hawthorne, NY, Nov. 2005. (Also submitted to the IEEE TC Journal).

[5] R. K. Balan, M. Satyanarayanan, S. Park, and T. Okoshi. Tactics-based remote execution for mobile computing. In *Proceedings of the* 1^{st} *International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 273–286, San Francisco, CA, May 2003.

[6] J. Cheng, R. K. Balan, and M. Satyanarayanan. Exploiting rich mobile environments. Technical Report CMU-CS-05-199, Carnegie Mellon University, Pittsburgh, PA, Dec. 2005.

[7] Id Software. QUAKE 2 source code. http://www.idsoftware.com/business/techdownloads/, Apr. 2002. (Version 3.21).

[8] D. Narayanan and M. Satyanarayan. Predictive resource management for wearable computing. In *Proceedings of the* 1^{st} *International Conference on Mobile Systems, Applications, and Services (MobiSys)*, San Francisco, CA, May 2003.

[9] T. Riker. BZFLAG source code and online documentation. http://www.bzflag.org/, June 2003. (Version 1.10).

[10] R. Sailer, L. van Doorn, and J. P. Ward. The role of TPM in enterprise security. Technical Report RC23363(W0410-029), IBM Research, October 2004.

[11] J. P. Sousa. *Scaling Task Management in Space and Time: Reducing User Overhead in Ubiquitous-Computing Environments*. PhD thesis, School of Computer Science, Carnegie Mellon University, May 2005.

[12] J. P. Sousa, R. K. Balan, V. Poladian, D. Garlan, and M. Satyanarayanan. Giving users the steering wheel for guiding resource-adaptive systems. Technical Report CMU-CS-05-198, Carnegie Mellon University, Pittsburgh, PA, Dec. 2005.

[13] M. Toennies. DAIMONIN source code. http://daimonin.sourceforge.net/, Sept. 2003. (Version 0.96alpha1).

[14] B. S. Woodcock. Graphing the growth of mmogs. http://www.mmogchart.com/, Mar. 2004.