Research Statement

Mohan Rajagopalan

My research aims to combine aspects of operating systems, program analysis, and compiler optimization, areas of systems research that have traditionally evolved orthogonally and in relative isolation. In particular, my dissertation work has focused on developing new approaches to system optimization that are both holistic and that address attributes such as dependability in addition to performance. The key distinguishing feature of this work is its use of traditional compiler techniques in novel contexts, an approach that not only facilitates automation but also helps ensure the correctness of the resulting system.

Holistic System Optimization

Techniques for optimizing system software have evolved along two distinct paths, targeting either individual applications (compiler concerns) or policies that affect global system behavior (operating system concerns). This traditional partitioning imposes significant restrictions on the types of optimizations possible, their effectiveness, and the ability to automate this process. For example, even though the execution of a program almost always involves interactions across multiple address spaces, optimizations that span address-space boundaries are rare.

My dissertation shows that such problems can be addressed and systems optimized more effectively by using a holistic approach that combines the fine-grain low-level information obtained through program analysis techniques such as static analysis and profiling, with coarse-grained high-level system execution semantics such as operating system policies and mechanisms. A key component of this research has been the use of proven compiler techniques to ensure transparency, automation, and correctness. Specifically, in this research I have developed a new set of compiler optimizations targeting event-based programs [2]; reduced the cost of system calls through an automatic compiler-assisted approach, *system call clustering* [1]; increased dependability by protecting against intrusions through a novel operating system primitive, the *authenticated system call* [3]; and developed a new binary rewriting framework called *Charon* that is capable of transforming operating system kernels. Each is now described in turn.

Event Optimizations. Events are increasingly used as a fundamental programming abstraction in a variety of contexts since they provide an extremely flexible structuring paradigm that can be used to realize richer execution semantics than thread and procedural-oriented styles. However, given the basic indirection between modules that raise and handle events, standard program analysis techniques have largely been ineffective in identifying connections between events and their handlers. As a result, event-based programming often has the potentially serious drawback of high execution overheads.

In this research, I have developed new compiler techniques for optimizing event-based systems. The underlying predictability exhibited by these programs is exploited to predict program behavior statically. Specifically, I developed a model for describing event-based execution, techniques for profiling event execution, and optimizations targeting event and handler sequences. Event execution across modules was profiled to identify commonly encountered events and event sequences, which were then optimized using static compiler techniques. Experimental results indicate that these optimizations are promising for improving the performance of a range of programs such as protocol stacks for applications such as H263-based video players, and applications based on X Windows.

System Call Clustering. Reducing the overhead of system calls can have a significant impact on overall system performance. Not only are they expensive—more than 20 times the cost of a regular procedure in terms of CPU cycles plus second level effects such as cache misses—they are also widely used as each program makes thousands of system calls in a typical execution. This combination of cost and ubiquity means that optimization of system calls, both individually and for a program as a whole, can potentially have a large impact on overall program performance.

I developed a new holistic optimization approach, *system call clustering*, that focuses on improving a program's entire system call behavior rather than simply reducing the cost of an individual call. The underlying premise of the approach is to replace groups of system calls with a single call implementing their combined functionality, thereby reducing the number of kernel boundary crossings. This transformation involves three steps. First, profiling is used to identify system call clusters—sequences of system calls that can be grouped together, potentially from non-consecutive statements and across function boundaries, and then executed in a single kernel crossing. Next, correctness-preserving compiler techniques such as code motion, function inlining, and loop unrolling are used to transform the program to maximize the size of the clusters given the semantics of the program. Finally, each sequence is replaced by a single call to a generic *multicall* mechanism that is implemented using kernel extension facilities. As a prototype, multicalls were implemented in Linux using loadable kernel modules, with the transformations realized as an optimization pass in the Pentium Link Time Optimizer (PLTO), a binary rewriting framework targetted at IA-32 binaries. In addition to significant performance improvements, desirable side-effects included reduced power consumption and improved cache behavior.

Authenticated System Calls. Intrusion tolerance has become an important area of systems research given that security incidents continue to occur with increasing frequency despite the increased attention being paid to computer security. System call monitoring is a technique for detecting and controlling compromised applications by checking at runtime that each system call conforms to a policy that specifies the program's normal behavior. Current implementations of such security mechanisms suffer from drawbacks such as unacceptably high performance overheads and the potential introduction of race conditions that introduce unwanted complexity.

I developed a new holistic approach to system call monitoring that combines the salient features of fine-grain sandboxing with the expressiveness of policy-based techniques. The basis of this approach is a new operating system primitive, the *authenticated system call* (ASC). An ASC is a system call augmented with extra arguments that specify the policy for that call and a cryptographic message authentication code (MAC) that guarantees the integrity of the policy and the system call arguments. This extra information is used by the kernel at runtime to verify the system call. The version of the application in which regular system calls have been replaced by ASCs is generated automatically by an installer program implemented using PLTO that reads the application binary, uses static analysis to generate policies, and then rewrites the binary with the authenticated calls. This approach has several advantages compared with existing systems, including significantly smaller overheads and policy generation that is easier, portable, and exhaustive. Further extensions to this research are directed at authenticating control flow and using dynamic analysis to increase the precision of generated policies.

Charon: Binary Rewriting of OS kernels. Even though binary rewriting is a promising technique for transforming and improving operating systems, existing systems fail to cope with the complexities involved. For example, unlike regular application binaries, kernel binaries have a significant amount of data embedded within executable sections, include implicit addressing constraints, and have unusual instruction sequences that are a result of hand-coded assembly and self-modifying code.

I developed a new binary rewriter, Charon, targeted at transforming operating system kernel binaries. Built on PLTO, this work introduces a number of novel mechanisms for kernel analysis and transformation, such as an indigenous type-based disassembly algorithm and a new extended control flow graph abstraction. As a preliminary application of Charon, we are looking at automating kernel customization. Specifically, the goal is to specialize a given kernel for a given set of requirements, such as a fixed set of programs and hardware specifications, in order to reduce the memory footprint for embedded devices.

Future Directions

I would like to explore applications of these holistic approaches to system analysis and optimization in broader contexts. Specifically, I would like to investigate new programming abstractions and richer execution models that can be used to structure, understand, and adapt systems better. In this section, I describe three ideas that demonstrate this vision.

Automatic OS Specialization. Charon is a powerful tool that provides the ability to both analyze and transform operating system kernels. An immediate extension to memory-footprint reduction would be to explore the concept of a $\eta - kernel$ in which a combination of static and dynamic program analyses and compiler optimizations are used to automate the specialization process, possibly on a per deployment basis. In this research, I would like to explore different aspects of customization, such as improving performance, reliability and security. As a simple illustrative example, one can imagine the use of compiler techniques to improve the overall performance of the system by, for example, introducing specialized versions of frequently used system calls, moving functionality into and out of the kernel in a provably safe manner, and providing hints to improve scheduling and resource allocation.

Improving OS Interfaces. While the structure and requirements of operating systems have undergone drastic evolution over the years, surprisingly, the interfaces and abstractions have remained largely unchanged. I would like to explore the possibility of developing new interfaces and richer abstractions, which would not only be directed at enhancing attributes such as the reliability, scalability, and performance of commodity operating systems, but also make them easier to structure, analyze, and optimize. A simple example would be that of enhancing the OS-application interface so that applications can inform the operating system about their quality of service requirements. This would allow both the OS and application to adapt to various events such as changes in network characteristics. Another would be seeing how the basic abstractions, eg. processes, can be extended to convey meta data such as security policies or program execution traces that can be used at runtime to monitor and adapt the behavior of the underlying OS.

Dependability. Different dependability attributes, including security, reliability and timeliness, can benefit from a holistic approach. For example, fine-grain information obtained through program analysis can be used to guide the replication of distributed objects or to develop new dependability mechanisms for operating systems. Another related idea would be improving the precision of system-wide security policies through static and dynamic analysis, to deal with for example, self-modifying code. Finally, I am interested in exploring dependability issues that are specific to binaries or generated code, a topic of growing importance as software is increasingly composed of independently developed components. An illustrative example of such a task would be the use of inter-address space profiling to check for race conditions across modules or module compatibility in large applications.

References

[1] M. Rajagopalan, S. K. Debray, M. Hiltunen, and R. Schlichting. System call clustering: An automated approach to system call optimization. In preparation.

[2] M. Rajagopalan, S. K. Debray, M. Hiltunen, and R. Schlichting. Profile-directed optimization of event-based programs. In *Proc. ACM SIGPLAN '02 Conference on Programming Language Design and Implementation (PLDI-2002)*, pages 106–116, June 2002.

[3] M. Rajagopalan, M. Hiltunen, T. Jim, and R. Schlichting. Authenticated system calls. In *Proc. IEEE International Conference on Dependable Systems and Networks (DSN-2005)*, June 2005. to appear.