GSRC 2006

Proceedings of

The First Annual Graduate Student Research Conference

October 13, 2006
Santa Barbara, California

Sponsored by The Department of Computer Science
University of California, Santa Barbara
http://www.cs.ucsb.edu

GSRC 2006

Proceedings of

The First Annual Graduate Student Research Conference

October 13, 2006
Santa Barbara, California

Sponsored by The Department of Computer Science
University of California, Santa Barbara
http://www.cs.ucsb.edu

Message From the Conference Chairs

Dear Computer Science Department,

It is with tremendous excitement that we present to you the proceedings of the first annual Computer Science Graduate Student Research Conference. Our department shines in two areas: our commitment to technical excellence, and our welcoming, cordial, and collaborative research community. Our hope is that this conference provides another opportunity to bring together students and faculty, both academically and socially, and strengthen the ideals that make this department dear to me. And we encourage future generations of graduate students to carry on this tradition with conferences like this.

Within these pages you will find ten abstracts describing new, upcoming research projects from our department's graduate students. Papers for this conference were selected first and foremost on their technical quality, but were also reviewed for clarity and presentation style. The Program Committee and the chairs are very pleased with the quality and the diversity of the research contained here. We present these to educate the reader about ongoing research in the department, encourage communication and collaboration, and stimulate new projects. Also, we hope to provide the authors with encouragement, feedback, and support on the research projects that will hopefully carry them through their degrees here.

There are many thanks necessary to those that helped make this conference a reality. First, we owe tremendous thanks to the Program Committee, who dedicated time and attention to selecting quality material for these proceedings. Our department staff–Shelly Vizzolini, Amanda Hoagland, and Greta Carl-Halle–also help to plan and locate rooms for this event. Finally, we want to thank Fred Chong for giving us the opportunity to organize this, and our advisors, Rich Wolski and Kevin Almeroth, for tolerating yet another diversion.

Matthew S. Allen and Allan Knight
Committee Chairs

Contents

GSRC 2006 Program Committee Members

Committee Chairs:	Matthew S. Allen, *Distributed Systems*
	Allan Knight, *Education Technology*
Faculty Advisor:	Fred Chong
Committee Members:	Stephen DiVerdi, *Computer Vision*
	Sotiria Lampoudi, *Computational Science and Engineering*
	Darren Mutz, *Network Security*
	Sara Woodworth, *Theory*
	Kimaya Mittal, *Wireless Networking*
	Viral Shah, *Scientific Computing*
	Priya Nagpurkar, *Compilers*
	Ryan Dixon, *Architecture*
	Ahmed Metwally, *Database Systems*
	Vebjorn Ljosa, *Bioinformatics*
Administrative Support	Greta Halle *Student Affairs Manager*
	Amanda Hoagland *Graduate Program Coordinator*
	Sandy Jacobs *Undergraduate Program Coordinator*

GSRC 2006 Events Schedule

10:00	**-**	**10:10:**	**Setup,** *Opening Remarks*
10:10	**-**	**10:20:**	**Matthew Allen,** *Opening Remarks*
10:20	**-**	**10:40:**	**Arsany Sawires,** *XML View Maintenance*
10:40	**-**	**11:00:**	**Qingqing Yuan,** *Online Quantum Algorithms*
11:00	**-**	**11:20:**	**Ted Huffmire,** *Policy-Driven Separation for Systems-on-a-Chip*
11:20	**-**	**11:40:**	**Amit Jardosh,** *IQU: Practical Queue-Based User Association Management for WLANs*
11:40	**-**	**12:00:**	**Lamia Youseff,** *Application Specific Linux (ASL)*
12:00	**-**	**12:30:**	**Pizza Lunch**
12:30	**-**	**1:30:**	**Josep Blanquer,** *Invited Speaker*
1:30	**-**	**1:40:**	**Break**
1:40	**-**	**2:00:**	**Justin Muncaster,** *Classification of Abnormal Activities in Videos*
2:00	**-**	**2:20:**	**Greg Banks,** *Application Testing and Analysis for Security*
2:20	**-**	**2:40:**	**Ye Wen,** *Simulation of Large Scale Sensor Networks*
2:40	**-**	**3:00:**	**Jonathan Ventura,** *A Sketch Interface to Aid 3D Scene Reconstruction*
3:00	**-**	**3:20:**	**Stephanie Taylor,** *Analyzing the Phase Behavior of the Circadian Clock in Arabidopsis thaliana*
3:20	**-**	**3:30:**	**Allan Knight,** *Closing Remarks*

XML View Maintenance

Arsany Sawires Prof. Divyakant Agrawal Prof. Amr El Abbadi

Distributed Systems Lab (DSL)
Department of Computer Science
University of California Santa Barbara (UCSB)
{arsany,agrawal,amr}@cs.ucsb.edu

1. INTRODUCTION

Caching frequently accessed items is a well-known technique in both hardware and software computer systems. The main goal is to improve the system performance by avoiding some communication and/or processing costs. In this paper we focus on caching the results of frequent queries in data management systems, in hope for answering future queries from the cached results. Answering queries from caches has proved to be a very successful technique for improving performance of data management systems. The approach is commonly known in the world of data management as *View Materialization*. If a query (view) is known to be frequent, then its result is computed from the base (source) data and cached (materialized) so that some future queries can be answered from the cache. This technique can significantly save processing and/or communication costs. The savings in processing cost are mainly due to the fact that the size of the cached result is usually orders of magnitude less than the size of the base data, and thus the cache computations run much faster. As for the communication cost, significant savings can be achieved in data-centric distributed systems, such as web databases, where caching in a middle-tier (between the server and the client) can help avoiding (or shortening) the trip of the query from the client to the data server and, more importantly, the trip of the query result back from the server to the client. Figure 1 illustrates the use of materialized views.

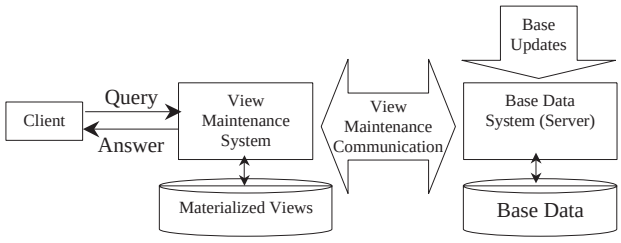


Figure 1: Using and Maintaining Mat. Views

For a motivating example, assume a query that you have issued against Amazon.com asking for all books by author "X" since 1990. Now assume that you, or another client, are asking for all the books by the same author "X" since 2000. Obviously, a cache holding the result of the first query can efficiently answer the second query due to savings in processing and in communication as explained above.

The technique of view materialization has raised several interesting research-worthy questions in the field of data management. Given a query Q and a cache holding the result of a previous query (view) V, is it possible to answer Q using V? If yes, then how to rewrite Q into Q' such that when Q' is computed using the the cache V, we get the same (or almost the same) answer as if we compute Q using the entire base data, e.g. the entire Amazon.com database? This problem is known as *Rewriting Queries Using Views*. Another basic problem is that of *View Selection*, namely, what are the views (queries) that we should select to be materialized such that the entire system performance is optimized?

In this paper, we do not discuss more about the two problems of rewriting queries and view selection, rather, we focus on a third fundamental problem raised by the approach of view materialization. We focus on the problem of maintaining the consistency of the cached results when updates take place at the base data. Back to our Amazon.com example: assume that after caching the result of the first query in a middle-tier, Amazon.com decided to offer a 1-day 20% sale on the books by author "X". Now, the information in the cache is inconsistent with the base data since they reflect higher prices than the actual current prices. This inconsistency is not in favor of neither the server (Amazon.com) which could lose business, nor the client which could miss the sale opportunity. Thus, the cache should be updated according to the base updates in order to restore the cache consistency. The degree to which inconsistency can be tolerated depends on the application. The process of restoring the cache (view) consistency is known in the data management world as *View Maintenance*.

A naive approach to view maintenance is to recompute the cached results whenever updates take place at the source data. The processing and communication costs of this naive strategy is usually prohibitive, especially when base updates happen frequently. Very often, the base updates do not cause any change in the cached view results, i.e. the updates are irrelevant to the views, and thus any view recomputation is redundant work. Even when an update affects the cached view result, it is usually much more efficient to incrementally maintain the view, if possible, rather than naively recomputing the entire view result. Hence, the problem of *Incremental View Maintenance* has received significant attention in the database research community.

The traditional model for data management systems is the well-known relational model. Recently, the XML tree-based data model has become a de-facto standard for representation and exchange of data. The XML model is gaining pop-

ularity mainly because it fits very well in the new world of data management where integration and inter-operation between heterogeneous systems have become real needs. Given this, the data management community started revisiting the classical problems, such as View Maintenance, with the XML data model. In this paper, we briefly abstract some recent work that we have done at UCSB regarding the problem of maintaining materialized views defined over XML base data.

Views (queries), and base updates are defined using XML-specefic data manipulation languages. The World Wide Web Consortium (W3C), which is responsible for issuing standard specifications for the WWW, has recommended a tree navigation language for XML documents (trees), the language is known as XPath. Another language, XQuery, which extends the power of XPath is expected to become a W3C recommendation sometime this year (2006). In our work so far we have focused on XPath as a basic XML query language. For a simple example of an XPath expression, consider the following expression (some syntax simplification is applied here):

$$doc(library.xml)/book[author = "X"][year >= 1990]/title$$

This expression has 3 steps: the first specifies the XML document (root of a tree) as "library.xml" , the second selects books satisfying some criteria, and the third selects the titles of the books selected in the second step. The second step has two selection predicates (criteria): the author is "X", and the year is 1990 or later. In general, an expression can have any number of steps and predicates.

We have identified two main variations of the problem of maintaining views defined using XPath expressions; Section 2 briefly describes these variations (models), and summarizes the solutions that we have proposed. Section 3 discusses future research directions for the problem.

2. TWO MODELS

Figure 1 shows that the base data system and the view (cache) maintenance system need to communicate with each other in order to maintain the consistency of the materialized views. As mentioned above, we have identified two variations (models) of the view maintenance problem. Intuitively, the main distinction between the two models is the degree of "coupling" between the view maintenance system and the base data system. The degree of coupling between these two systems determines the type (degree of specificity) of information that can be communicated between the two systems. If the systems are "tightly coupled", they can exchange detailed (very specific) information to facilitate the view maintenance process. On the other hand, if they are "loosely coupled", then the degree of specificity of the communicated information is limited because it has to comply with the universal standard specifications, such as the XPath language.

The tight-coupling model is suitable for situations where the base data system and the view maintenance system are actually the same; this can be the case when the views are materialized within the base data system itself to improve the performance of query processing. On the other hand, the loose-coupling model has become a reality in today's world where the WWW technologies have allowed heterogeneous and autonomous systems to inter-operate through universally standard protocols and specifications.

In [2], we have proposed a solution for the tight-coupling model. Given a base update, this solution can always incrementally maintain the materialized views very efficiently, relative to the naive view recomputation approach. To achieve this result, we have assumed the following tight-coupling assumptions:

1. Very specific information, known as *update paths*, about each base update is reported by the base data system to the view maintenance system. Particularly, the exact locations of the updated nodes in the XML tree are reported.
2. The view maintenance system can issue queries to the base data system such that the execution of these queries starts at nodes other than the tree root. This assumption is essential for the efficiency of the incremental maintenance solution.
3. The results returned by the base data system in response to the view maintenance queries include ids of the XML tree nodes.

To enable efficient incremental maintenance, the solution we provided for this model maintains some auxiliary information, known as *result paths*, in addition to the materialized views. In this context, our solution enjoys the following desirable property: the size of the auxiliary information is proportional to the size of the materialized views. The main conclusion of this work is that using update paths and result paths, we can achieve efficient and scalable incremental XPath view maintenance.

In [1], we have relaxed the three assumptions mentioned above since none of them is valid in loosely coupled settings. In the new model, the view maintenance system uses only the following information: (1) the update expression, rather than the specific update path, (2) the view expression, and (3) the materialized view result (which does not include any node ids). This model is suitable for loosely-coupled systems. We have shown that incremental maintenance is not always possible under this model. Thus, maintaining the consistency of the views requires frequent view recomputations. Hence, our goal in this work is to reduce the frequency of view recomputation. We do this by detecting cases where a base update is irrelevant to a view, and cases where a view is self maintainable (maintainable without base queries) under a base update. The experimental results show the effectiveness of the proposed approach in reducing view recomputations.

3. LOOKING FORWARD

Some issues remain open for future work: we would like to extend the data and query model to handle order in XML documents; we would also like to explore the possibilities of using schema information, if available. For more details, please see [2, 1].

4. REFERENCES

[1] Arsany Sawires, Junichi Tatemura, Oliver Po, Divyakant Agrawal, Amr El Abbadi, and K. Selçuk Candan. Maintaining xpath views in loosely coupled systems. In *VLDB*, 2006.

[2] Arsany Sawires, Junichi Tatemura, Oliver Po, Divyakant Agrawal, and K. Selçuk Candan. Incremental maintenance of path expression views. In *SIGMOD Conference*, 2005.

Online Quantum Algorithms

Qingqing Yuan
qqyuan@cs.ucsb.edu

Wim van Dam
vandam@cs.ucsb.edu

ABSTRACT

In recent years, online monitoring of data streams has emerged as an active topic in data management field. A lot of research has been focused on finding the most frequent items in the data streams. In this paper, we will try to use quantum mechanisms to develop online algorithms for frequent items finding. The main contribution of our work is that we prove the lower bound of space requirement for quantum online algorithms that determine the most frequent item.

1. INTRODUCTION

Recently, much attention has focused on the theory of *online monitoring* of *data streams*, which is relevant in many domains, including web-applications, network security, sensor monitoring, etc. In [2], the authors summarize that the data stream model differs from the conventional stored relation model in the following ways: The data elements are processed online; The data streams are potentially unbounded in size; and each element is processed only once.

One of the most basic problems on a data stream is that of finding the most frequent items in the stream. A lot of research has been done on estimating the most frequent items in a data stream using limited storage space. A well known lower bound of $\Omega(n)$ is given by Alon et al [1], where n is the size of the alphabet of items.

At the other end of the spectrum, since the mid 1980's, quantum computation and quantum information are extensively studied, which yields many new and exciting capabilities for information processing and communication. For instance, while the best-known factoring algorithms for a classical computer run in exponential time, Shor's quantum algorithm can factor an n-bit integer in $O(n^3)$ time [7]. Recently the space complexity of quantum algorithms has been extensively studied. le Gall[5] demonstrates a special language which can be recognized by an online quantum Turing machine with bounded-error which incurs an exponential decrease in the space complexity in comparison to using the classical Turing machine to solve the target problem.

In this work, we will try to use quantum superpositions to store the information from the data stream, and extract useful result from these superpositions online. We're hoping to use less qubits than classical bits for online algorithms. The main result of this paper is that we prove the lower bound of space complexity for online quantum algorithms of estimating the most frequent item is $\Omega(n)$.

2. CLASSICAL ONLINE ALGORITHMS

We often talk about the problem of finding the most *frequent elements* or *top-k elements* in a stream. Given an alphabet, A, let $S = (s_1, s_2, \ldots, s_m)$ be a a stream of size m, where $s_i \in A$. A *frequent element* s_i is an element whose frequency exceeds a user specified support $\lceil \phi m \rceil$, where $0 \leq \phi \leq 1$; whereas the *top-k elements* are the k elements with highest frequencies. Many algorithms have been proposed to solve this problem. An overview of the algorithms is shown in Figure 1. The main goal of the algorithms is to make the local memory as small as possible. The methods can be mainly categorized into *counter-based*[4] and *sketch-based*[6] algorithms.

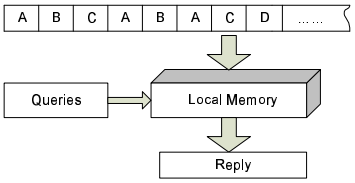


Figure 1: A schematic overview of an online algorithm receiving queries on a data stream

Counter-based Algorithms keep an individual counter for each element in the monitored set, which is a subset of A. If the current element is in the monitored set, its counter is updated; otherwise it is ignored, or some algorithm-dependent actions are taken.

Sketch-based algorithms, instead, provide frequency estimation for all elements by using bitmaps of counters. Specifically, each new observed element is hashed into the whole counter space using a family of hash functions. The hashed counter is updated for every hit of the new observed element. The hash collisions inevitably incur accuracy loss of the final reported frequency of each element.

In general, counter-based algorithms process each item more efficiently. However, the accuracy likely depends on the element order in a data stream since it always involves a specific strategy of choosing a subset of the elements to be monitored. On the contrary, sketch-based algorithms are able to keep track of all the elements using a hash mapping, and hence they are independent on the element order in a data stream. The major disadvantage of sketch-based algorithms lies in the expensive computation of processing each item. Moreover, the accuracy will be degraded significantly due to hash collisions.

The exact solution to the original frequent items problem has impractical space requirements. Alon et al.[1] give the $\Omega(n)$ lower

bound on the space complexity of any algorithm for estimating the most frequent item in an arbitrary data stream.

3. ONLINE QUANTUM ALGORITHM

3.1 A Model of Quantum Online Algorithms

Let $X = (x_1, x_2, \ldots, x_m)$ be a sequence of elements, where $x_i \in A = \{1, 2, \ldots, n\}$. Consider a quantum computer with an initial state ρ_0, where ρ_0 is the memory of s qubits for our quantum computer. We define n transformations: $\{T_{x_i} : x_i \in A\}$, and each of them is a transformation acting on s-qubits. Thus, the machine can perform different transformations according to the content of the current data item x_i. Once a piece of data x_i is available, the machine is evolved from state ρ_i to ρ_{i+1}.

The evolution of the quantum computer with respect to the whole data stream can be described as following:

$$\rho_0 \overset{T_{x_1}}{\longmapsto} \rho_1 \overset{T_{x_2}}{\longmapsto} \cdots \overset{T_{x_m}}{\longmapsto} \rho_m$$

As T_{x_i} can be a non-unitary transformation, state ρ_i could be a *mixed* state.

3.2 Lower bound of space requirement

Let $f_i = |\{j : x_j = i\}|$ represents the number of occurrences of element i in the data stream, and the task of estimating the most frequent alphabet can be defined as below: $F_\infty = \max_{1 \le i \le n} f_i$.

In this sections, we focus on the lower bound for the space complexity of online quantum algorithm approximating F_∞. As in [1], the lower bound can be obtained after reducing the problem to an appropriate communication complexity problem.

Before presenting the space bound for approximating F_∞, we first introduce some preliminary knowledge regarding the *quantum communication complexity* with respect to a function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, introduced by Yao[8]. Consider two parties, A and B, with unlimited power to compute the value of a Boolean function $f(x, y)$, where x and y are binary vectors of length n, party A possesses x and party B possesses y. To compute the value, the parties are allowed to exchange qubits with each other. At the end of the communication, party B outputs the value of $f(x, y)$. The complexity is measured by the number of qubits exchanged in the worst case. One of the most prominent functions used in above-mentioned algorithm is the disjointness function $DISJ_n$. Let $DISJ_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ denote the Boolean function(called the *Disjointness function*), where $DISJ_n$ is 1 if and only if two subsets of $\{1, 2, \cdots, n\}$ whose characteristic vectors are x and y intersect.

It is known that the communication complexity of $DISJ_n$ for one-round protocols is $\Omega(n)$[3]. Inspired by their work, we try to prove the lower bound of the space requirement for approximating F_∞.

Theorem: For fixed probability $\epsilon > 0$, any online quantum algorithm that approximates F_∞ of sequences X of $2n$ elements from an alphabet $A = \{1, ..., n\}$ within $[2/3F_\infty, 4/3F_\infty]$ with success probability at least ϵ requires $\Omega(n)$ memory of qubits.

Proof: We prove this theorem by reducing the problem to an appropriate communication complexity problem. Given an algorithm as above that uses s qubits memory, we describe a communication protocol using only s qubits communication.

Two strings x and y are described in the communication complexity problem before. Let $|x|$ and $|y|$ denote the numbers of 1's of x and y. X is the sequence of length $|x|+|y|$, which contains all members of the subset of A whose characteristic vector is x, followed by all members of the subset of A whose characteristic vector is y.

Party A, with initial state ρ_0 and string x, runs the quantum algorithm on the first $|x|$ members of X for estimating F_∞. The memory state evolves into $\rho_{|x|}$. And then, it passes the content of the memory to party B knowing y, which continues processing the next $|y|$ elements. Finally, the memory state reaches $\rho_{|x|+|y|}$. By measuring this final state, we obtain the value of Y. According to the result, Party B outputs "disjoint"(0) if and only if Y is smaller than $4/3$, otherwise "not disjoint"(1). Obviously, this Y value should be correct with a probability above $1 - \epsilon$, because the true value of F_∞ is 1 if the sets are disjoint, and 2 otherwise.

Generalized by the result of quantum communication complexity which has already been proven, at least $\Omega(n)$ qubits are required for one-round communication. So the theorem holds.

4. FUTURE WORK

In section 3.2, we proved that the lower bound of the space complexity for computing the most frequent item in the data stream is $\Omega(n)$, which is the same complexity as in the traditional algorithms [1]. This result indicates that the quantum mechanism does not improve the lower bound towards the exact solution of finding the most frequent item. However,in classical settings, we can reduce the space requirement substantially by approximating the original problem. Under various approximation conditions, researchers obtained different lower bounds of the space complexity and accuracy loss. Therefore, the future work, as well as our hope, lies in the possibility of reducing space requirements and improving the accuracy using smart relaxation techniques under the quantum mechanism.

5. REFERENCES

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *STOC'96*, 1996.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. *Invited paper in PODS 2002*.

[3] H. Buhrman, R. Cleve, and A. Wigderson. Quantum vs. classical communication and computation. *arXiv:quant-ph/9802040v2*, Mar 1998.

[4] E. D. Demaine, A. Lpez-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. *Proceedings of the 10th Annual European Symposium on Algorithms*, pages 348–360, 2002.

[5] F. le Gall. Exponential separation of quantum and classical online space complexity. *SPAA'06*, pages 67–73, July- Aug 2006.

[6] A. Metwally, D. Agrawal, and A. E. Abbadi. Efficient computation of frequent and top-k elements in data streams. In *ICDT*, pages 398–412, 2005.

[7] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *FOCS*, pages 124–134, 1994.

[8] A. Yao. Quantum circuit complexity. *FOCS*, pages 352–361, 1993.

Policy-Driven Separation for Systems-on-a-Chip

Ted Huffmire
ArchLab
huffmire@cs.ucsb.edu

Tim Sherwood
ArchLab
sherwood@cs.ucsb.edu

ABSTRACT

Many embedded applications are implemented on systems-on-a-chip (SoCs) that contain multiple interacting components that may operate at different trust levels and clearance levels. These components typically share resources such as memory. For example, a SoC might contain two CPU cores, a Crypto core, and a DSP core, all sharing memory. A similar example is a multi-core processor in which different cores operate at different levels of security. Our goal is to make sure that cores share resources nicely by developing *efficient* hardware mechanisms.

Providing separation among these modules is a crucial security primitive. Separation is the enforcing of the legal sharing of a resource such as memory among multiple cores. Suppose Alice works for Company A, and Bob works for Company B. Alice's core must not be able to obtain the data of Bob's core and vice-versa.

The easiest way of providing separation is to implement each core on its own physically distributed chip. However, physical separation makes it impossible to realize the cost savings of increased integration. Another option is to implement the separation in software. Separation kernels [1] use software virtualization to provide each application with the same level of isolation it would have with its own physically distributed processor. The disadvantage of this approach is that there is extra overhead associated with providing virtualization. In addition, the design complexity of modern out-of-order microprocessors makes it difficult to implement separation kernels with a verifiable level of trust. Furthermore, since separation kernels are a software-based scheme, they will not work for embedded applications that lack code and a program counter.

Standard memory protection, which relies on a page table to prevent processes from accessing memory outside their assigned virtual address space, is not a workable solution for embedded systems either. In standard memory protection, if a process attempts to access memory outside this range, a page fault occurs, and the process is terminated. Since standard memory protection was never designed with rigorous security as a design requirement, it is inadequate for applications that need to be able to handle multiple levels of classified data. Moreover, many embedded systems do not have a disk, operating system, page table, or TLB. Mondrian memory protection [2] provides memory protection at the granularity of a word rather than a page, but it suffers from the same problem.

An alternative way of providing separation is to use a reference monitor that enforces the legal sharing of memory. It is possible to design a system in hardware in which every memory access must be approved as legal by a reference monitor that enforces a memory access policy that every core on chip must obey. This approach combines the cost savings of increased integration without the overhead of software complexity. Reference monitors are not new; in fact, they belong to the class of enforcement mechanisms known as execution monitors [3]. However, our formulation of reference monitors in the embedded domain is new.

Our approach uses a specialized compiler to translate a memory access policy expressed in a specialized language to a table format that can be represented by a memory tile. This memory tile can be loaded onto a SoC, and the reference monitor enforces the policy specified by this tile. Our language specifies which modules (subjects) have which access rights to which ranges of memory (objects).

Figure 1 shows an example of a simple isolation scenario. $Module_1$ and $Module_2$ are in separate compartments. $Module_1$ can only access (read from or write to) $Range_1$, and $Module_2$ can only access $Range_2$. This scenario would be expressed in our language as the following:

Access→ {$Module_1$,rw,$Range_1$}|{$Module_2$,rw,$Range_2$};

Policy→ (Access)*;

Our policy compiler, which was made with the help of Lex and Yacc, converts this policy to a regular expression. Next, the compiler converts the regular expression to an NFA, and then it converts the NFA to a minimized DFA. Finally, the DFA is converted to a table format that can be loaded onto a special memory tile located on the chip. Our reference monitor enforces the policy specified by the memory tile.

In addition to isolation, our policy language is powerful enough to express a variety of classic security scenarios, including both stateless (fixed) and stateful (transitional) policies. Our language-based approach provides flexibility because changing a policy is much easier than redesigning by hand a reference monitor expressed in a hardware description language.

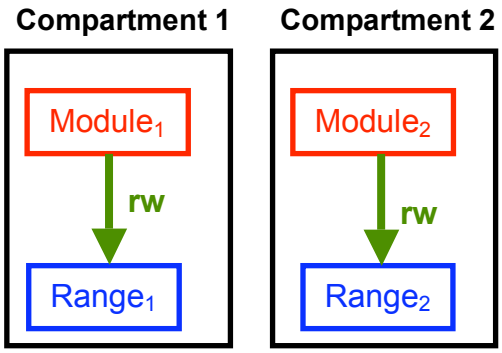


Figure 1: An isolation policy.

Systems often need to be designed to be responsive to external events. For example, if the system comes under attack, it makes sense to change from a less restrictive memory sharing policy to a more restrictive policy. A mechanism that can dynamically switch the policy enforced by the reference monitor is needed to achieve this goal. We call this mechanism a configuration manager.

The design of a configuration manager involves many interesting questions. Since area is at a premium, it is impossible to store every possible policy on a chip. We will make the following assumptions: that policy switching is rare, that we always switch to a more restrictive policy, and that the number of policies to switch between is small. All we need to do is have additional memory tiles for the different policies, and a simple hardware mechanism to tell the reference monitor which tile to use. The configuration manager will need to obey a *meta-policy* that specifies the conditions under which switching can occur and which core has the authority to perform a policy switch. A more sophisticated implementation will use a hot-swappable design in which one tile contains the active policy, and another tile is used for loading a new policy.

The fact that every memory access must go through the reference monitor raises an import issue. Multiple simultaneous memory references will have to be serialized, negatively affecting system performance. It is possible to overcome this problem by implementing several independent, distributed reference monitors that enforce the same policy. In the case of a stateless policy, all that is necessary is to duplicate the reference monitors, but in the case of a stateful policy, the distributed reference monitors will need to communicate with each other so that they are all in the same state.

Our policy language is designed with the restriction that it is only as powerful as a regular expression. This restriction exists because we are translating the policy to a DFA, which is equivalent to a regular expression. There may be some Turing-complete security policies that we cannot express in our language. Is there a way for us to modify our design in order to enforce a larger class of policies?

In any security technology, usability is a key concern. The reference monitor is only as good as the policy it is enforcing. Engineers need to be able to construct precise policies. Unfortunately, our memory access language is a relatively low-level language, and working in this language is prone to error. We must design a higher-level language for expressing concepts such as isolation and controlled sharing, and we must design a compiler to translate this high-level language into our low-level memory access language.

In addition to being usable, good security technology also must be low-cost. Our policy compiler can generate reference monitors that are efficient in terms of area and cycle time, but we must analyze the overall system performance of a realistic embedded application that uses our reference monitor approach. We plan to apply phase classification, which has been successful at analyzing systems in the microprocessor domain, to the embedded domain. Phase analysis classifies the repeating behavior of computer programs, and it enables more accurate simulation. Since phases repeat, it is unnecessary to perform the detailed simulation of a phase when it reoccurs. We plan to leverage this to our advantage in the quantitative analysis of our security methods.

Building such a realistic embedded application will force us to confront several issues related to the architecture of our security methods. For example, we must make sure that modules cannot spoof their identities and therefore confuse the reference monitor. We must also figure out how to make sure that every memory access goes through the reference monitor. We must determine the interface between the cores and the memory controller, and we must figure out how the reference monitor will connect to the memory controller. In addition, circuit elements can have values of 0, 1, X (undefined), and Z (high impedance line). In fact, there are around a dozen different values that have been discovered. However, security works in the binary domain (0 or 1). These different values (X, Z, etc.) could affect the security of any hardware security approach.

REFERENCES

[1] John Rushby. Design and Verification of Secure Systems. *ACM Operating Systems Review*, Vol. 15, No. 5, pp. 12-21, December 1981.

[2] E. Witchel, J. Cates, and K. Asanovic. Mondrian memory protection. In *Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, San Jose, CA, October 2002.

[3] Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1), February 2000.

IQU: Practical Queue-Based User Association Management for WLANs

Amit P. Jardosh, Kimaya Mittal, Krishna N. Ramachandran
Elizabeth M. Belding, and Kevin C. Almeroth
MOMENT Lab and NMSL, Department of Computer Science, UC Santa Barbara
{amitj,kimaya,krishna,ebelding,almeroth}@cs.ucsb.edu

[1]WLANs are indispensable for providing Internet access to users at locations such as universities, corporate offices, conferences, airports, and coffee shops. Many of these environments often experience flash crowds, which we define to be a sudden surge in the number of users simultaneously attempting to access the WLAN. When flash crowds occur, WLANs are likely to suffer from destructive interference, excessive channel load, and unsustainable packet processing at access points (APs). These conditions lead to a plethora of problems, such as a deterioration in network throughput, heavy packet loss, intermittent connectivity, overwhelmed APs, and sometimes, a network collapse.

To verify these claims, we present two case studies of operational WLANs that experienced the aforementioned problems. The two WLANs each consisted of over 100 APs and more than 1000 simultaneous users, deployed at recently held 62^{nd} and 64^{th} Internet Engineering Task Force (IETF) meetings [7]. In the first case study, a high concentration of users in adjacent rooms led to frequent packet collisions and detrimental interference. As a result, users experienced unusably low throughputs. Figure 1 shows the per-user throughput of each user during a one-second interval versus the number of instantaneous users during the same one-second interval. We observe from the figure that as the number of active users increases from 1 per second to 80 per second, the per-user throughput decreases significantly. In the second case study, users failed to establish associations with any APs due to either frequent packet collisions or excessive, unsustainable packet processing at the APs. The repeated association attempts made by users resulted in high control packet overhead, compounding the problem. The channels and the APs could not sustain such heavy workloads. The result was sparse or no connectivity for users in the network and eventual network collapse.

The connectivity and usage problems experienced by users at these events are not unique. Similar problems often occur in other scenarios, particularly those that are prone to high user concentrations, such as conferences and conventions. We predict that, as the popularity of WLANs continues to increase, these problems will become even more frequent and widespread and WLANs will have a greater need to handle flash crowds and large user concentrations.

As a result, an effective solution to manage a large number of users in a WLAN is imperative. The solution should not only avoid network breakdown, but also ensure connectivity and high user throughput. Several approaches to manage heavily loaded WLANs have been presented and evaluated in previous work. These approaches can be classified into four categories: *over-provisioning*, *selective dropping* [3], *load balancing* [2, 4, 8] and *traffic shaping* [5, 10]. Each category has its benefits and can marginally improve performance during a flash crowd. However, they each have drawbacks as well. Over-provisioning is expensive, inefficient and limited by bandwidth availability, while selective dropping may lead to starvation of some users. Balancing load among neighboring APs is of limited help when the total load is high enough to overwhelm all APs in the vicinity. Traffic shaping limits individual throughput in order to accommodate all users, and therefore, when the number of simultaneous users is very high, traffic shaping alone may result in unacceptably poor performance for most users.

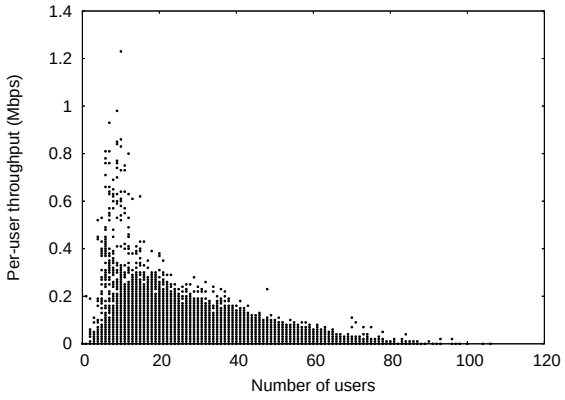


Figure 1: Per-user throughput.

WLANs that need to support a large number of users are thus in critical need of a practical and effective system to handle heavy loads and flash crowds. In this paper, we propose *IQU*, a practical queue-based user association management system for heavily loaded WLANs. The premise of user association management is to control the frequency and duration of user associations with the network when the number of users trying to access the network is greater than what the network can support. IQU maintains a queue of users requesting network access. Only as many users as can be simultaneously accommodated are granted access to the network. Any remaining users wait for admission in a queue. Admitted users are assigned periods of access, called *work-periods*, within which they can execute any network-related tasks. If the network is underloaded, the user queue will be empty and users can continue to access the network even after their work-period expires. In a heavy load situation, the expiration of the work-period causes the user to be disassociated from the network and placed back into the queue. A different user from the head of the queue is then admitted into the network. Users with network access are updated with their remaining work period so that they can plan their network-related tasks accordingly[2]. Similarly, users waiting in the queue are given esti-

[1]A complete version of this research has been recently accepted as a conference publication [6].

[2]Applications designed for disconnected operation [9] can also leverage this information.

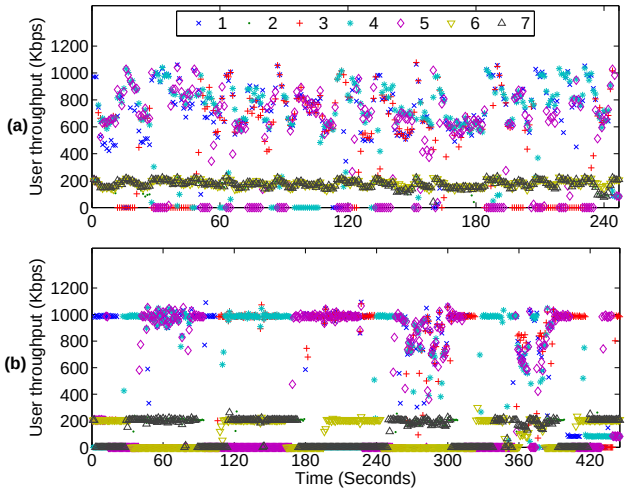


Figure 2: (a) Client throughput without IQU. (b) Client throughput with IQU.

mates of their wait time for network access and the duration of the work period they will be granted. Thus, unlike the scenarios presented in our case studies, there is no uncertainty about the availability of network access. This information prevents users from making repeated unsuccessful association attempts, thereby both reducing network overhead and considerably improving user experience.

IQU is a simple and powerful system for managing heavily loaded WLANs. However, IQU changes the basic access model to which today's WLAN users are accustomed. In heavily-loaded WLANs, IQU requires users to wait in a queue for access. Moreover, when access is granted, users must complete their network-related tasks within an alloted work period. This is a significant change from the current model of obtaining immediate access for unlimited durations of time. However, we believe that this change is inevitable in order to maintain good performance in a heavily-loaded WLAN. Moreover, we believe that the new model is intuitive and easy to understand. Users can be made aware of their assigned wait periods and work periods via a networking utility on the user's device. Note that the new access model may bring about unprecedented alterations in typical user behavior; for instance, users may generate traffic more quickly so that they can complete their tasks in the assigned work period.

To evaluate our system, we built a prototype of IQU and tested it on a testbed consisting of 8 Linux laptops (three IBM Thinkpads and five Toshiba Satellite laptops) equipped with Atheros chipset IEEE 802.11a/b/g wireless network cards. One laptop is configured to act as a wireless AP, while the remaining act as WLAN clients. The goal of this testbed is to convincingly demonstrate the practicality of an association management system such as IQU, as well as IQU's benefits in a real system. The AP and client laptops are placed within direct transmission range of each other. The wireless network cards are managed by the MADWiFi driver, which is a Linux kernel device driver module for Atheros-based WLAN devices [1]. We implement the IQU prototype by appropriately modifying this driver. For our experiments, we configure the wireless cards to use the IEEE 802.11b protocol and fix the data rate at 11 Mbps. We disable the RTS/CTS collision avoidance mechanism and MAC layer retransmissions. The impact of various IQU parameters on system performance is explored, and appropriate values for the parameters are identified.

We use our testbed to emulate a flash crowd and heavy load conditions. While three out of the seven clients maintain a UDP flow level of 200 Kbps in both directions combined, the remaining four clients simultaneously initiate UDP flow levels of 1 Mbps at the beginning of each experiment. Each UDP session lasts for four minutes. This traffic configuration emulates a flash crowd and heavy load conditions as observed in our case studies. Note that the four-minute durations do not include the time spent waiting in the user queue.

Figures 2(a) and 2(b) show the throughput achieved by each of the 7 clients without IQU enabled and with IQU enabled, respectively. In Figure 2(a) we observe significant losses and variations in the average individual throughput of all seven clients. This result clearly demonstrates the detrimental effect of a flash crowd and heavy load on network performance when IQU is not enabled. On the other hand, in Figure 2(b), we observe that IQU successfully controls the number of associated users such that, when admitted, each associated user's throughput close to the offered load.

Although IQU improves throughput during a flash crowd or high network load, the disadvantage is that it takes longer to service the clients in the network. This can be observed from the x-axis limits in Figures 2(a) and 2(b). The extent of increase in service time depends on the choice of parameter values and network traffic conditions. We argue that longer service times are an acceptable tradeoff for network administrators and users to avoid grossly unacceptable network performance or network collapse.

Due to its elegance and effectiveness, user association management has far-ranging implications as a tool for managing limited resources in sophisticated WLANs. It coalesces the benefits of over-provisioning, selective dropping, load balancing and traffic shaping, while avoiding their drawbacks. We believe that our work creates new directions for further research in this area. Different strategies can be explored for managing the user queue. Although we use a simple FIFO queue in this paper, priority-based queues may also be used to support different network access policies. Determination of the optimal number of users that may be permitted to simultaneously access the network and accurate estimation of user wait periods are other parts of this system that have potential for further exploration and research.

REFERENCES

[1] Multiband Atheros Driver for Wireless Fidelity (MADWiFi). http://madwifi.org.

[2] A. Balachandran, P. Bahl, and G. Voelker. Hot-Spot Congestion Relief in Public Area Wireless Networks. In *IEEE WMCSA*, Monterey, CA, Oct 2002.

[3] A. Barbaresi, S. Barberis, and P. Goria. Admission Control Policy for WLAN Systems based on the Capacity Region. In *IST Mobile Summit*, Dresden, Germany, Jun 2005.

[4] Y. Bejerano, S. Han, and L. Li. Fairness and Load Balancing in Wireless LANs Using Association Control. In *ACM Mobicom*, Philadelphia, PA, Sep 2004.

[5] C. Chiasserini and R. Rao. Performance of IEEE 802.11 WLANs in a Bluetooth Environment. In *IEEE WCNC*, Chicago, IL, Sep 2000.

[6] A. P. Jardosh, K. Mittal, K. N. Ramachandran, E. M. Belding and K. C. Almeroth. IQU: Practical Queue-Based User Association Management for WLANs. To appear in *ACM Mobicom*, Los Angeles, CA, Sep 2006.

[7] A. P. Jardosh, K. N. Ramachandran, K. C. Almeroth, and E. M. Belding-Royer. Understanding Congestion in IEEE 802.11b Wireless Networks. In *USENIX IMC*, Berkeley, CA, Oct 2005.

[8] A. Mishra, V. Brik, S. Banerjee, A. Srinivasan, and W. Arbaugh. A Client-driven Approach for Channel Management in Wireless LANs. In *IEEE Infocom*, Barcelona, Spain, Apr 2006.

[9] L. Mummert, M. Ebling, and M. Satyanarayanan. Exploiting Weak Connectivity for Mobile File Access. In *ACM SOSP*, Copper Mountain, CO, Dec 1995.

[10] M. Portoles, Z. Zhong, and S. Choi. IEEE 802.11 Downlink Traffic Shaping Scheme for Multi-User Service Enhancement. In *IEEE PIMRC*, Beijing, China, Sep 2003.

Application Specific Linux (ASL) *

Lamia Youseff Rich Wolski Chandra Krintz

Department of Computer Science
University of California, Santa Barbara
{lyouseff, rich, ckrintz}@cs.ucsb.edu

1. SUMMARY

Linux has emerged as the system-of-choice in academic and production scientific computing settings. A key limitation to the use of Linux for high-end cluster computing however is its potential performance impact on application execution, since Linux dictate general policies that do not promote the performance of high-end scientific applications.

In this abstract, we are presenting our Application Specific Linux (ASL), a customized Linux image that enhances the performance of the scientific applications. Our Research end-goal is a software system that automatically enables high performance scientific computing on commodity systems through application-specific customization and dynamic adaptation of the Linux OS. Our research is novel in that it combines the research done in OS specialization, extensibility and minimization as well as virtual machine monitors (VMMs) into a system that automatically customizes the kernel image for a single application run and is specifically focused on the application domain of scientific computing using high-performance clusters.

2. EXTENDED ABSTRACT

Recent advances in high-performance processor and network technologies are making clusters of workstation class computers cost-effective platforms that can support the next generation of scientific applications. Low per-unit cost, advances in computing and communication power, and the availability of Linux as a free, easy-to-use, and nearly standard operating system, make high-end computing with these systems accessible both to a very large developer base and to a wide range of users. As part of this evolution, Linux has emerged as a nearly ubiquitous, open-source operating system with a wide-range of readily available programming support tools and specialized libraries. It is currently the system-of-choice in academic and production scientific computing settings and as a result, many -if not the majority of- scientific programmers, being trained today are familiar with Linux as a development platform. Moreover, its ability to be used as a locally controlled, high responsive development environment has greatly increased the programmer productivity, hence the advancement of science.

A key limitation to the use of Linux for high-end cluster computing however is its potential performance impact on application execution [3]. Linux, like other general-purpose operating systems (GPOS) with commercial application, continues to evolve to support an enormous range of user requirements, application domains, and devices (everything from supercomputers to hand-helds). In contrast, scientific applications executing in clustered settings are frequently large, resource intensive, long-running, and use space-sharing to gain exclusive access to the machines they use through a batch system. They do not compete dynamically for processor and I/O resources like many other application domains. Therefore, the Linux OS includes many features and built-in policies that do not promote the performance of high-end scientific applications, which can retard the performance of scientific programs in high-end computing settings. In particular, scientific applications typically do not require the extensive support for fair resource sharing (since they execute in production space-shared, and not time-shared, environments) or quick response time (since they may not be interactive). None the less, the portability that Linux affords combined with the familiarity that its wide-spread popularity has bred make it a de facto standard operating system for clustered architectures.

In our Research, we are designing *A*pplication *S*pecific *L*inux (ASL) to enhance the performance of Linux for scientific applications[4]. The goal of our research is to investigate techniques that maintain the ease-of-use and cost benefits of Linux while enhancing the performance achievable by high-end scientific applications executing in large-scale cluster computing settings. To enable this, we are studying ways to automatically customize the Linux instance an application uses when it is running in a "production" (i.e., non-development or debugging) setting based on the specific needs of the application itself. We are also exploiting the exclusive processor access that batch-scheduling implements to relax or eliminate unneeded mechanisms that are designed to facilitate effective time-sharing, but which introduce unnecessary overhead in a space-sharing context.

We are using both runtime and compile-time approaches to customize the Linux instance used by each application. Each of which will allow the scientific programmer to use unmodified Linux as a local development and debugging environment and then to apply our techniques as an additional compilation step before initiating high-end cluster execution.

In order to enable Linux customization, our system consists of four logical customization phases. Static and dynamic application analysis is our system's first phase. The application as well as its potential coupling effect on the kernel is studied using Phases profiling techniques as well as kernel performance-annotated call graphs. The second phase is the static customization enhancements for the kernel image based on the outcome from the first phase, which is produced at the development site. For example, we studied in [7] the I/O pattern for MIT General Circulation Model

9*This work is sponsored in part by grant from the National Science Foundation (ST-HEC-0444412).

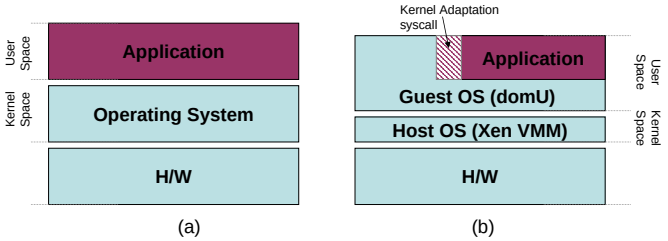


Figure 1: Deployment model for current scientific applications in (a) versus our deployment model using Xen VMM in (b).

(GCM) [5], which is a popular numerical simulation used by scientists to study oceanographic and climatologic phenomena. For this specific MIT GCM code I/O pattern, the sys_write() syscall was modified to enhance the application performance. The static customization phases furthermore include inlining the application code inside the kernel image to reduce the user to kernel space crossing overhead, as well as inlining some kernel modules and creating execution shortcuts in the kernel for enhanced performance of the common execution scenarios for the application, as shown in figure 1(b). The customized Linux image (including the application) is then shipped to the production environment.

The third phase takes place at the production environment, where ASL is deployed on a minimal virtual machine monitor (VMM), as shown in figure 1(b). This approach allows us also to introduce unsafe kernel customization while protecting volatile hardware resources (e.g. BIOS code). In order to ensure the feasibility of our approach, we have evaluated the performance implication of running HPC performance-oriented codes on virtual machines in [6]. We opt to deploy paravirtualized VM, in which the guest OS is aware of being virtualized. Paravirtualization is characterized by its minimal performance degradation on application execution, relative to full virtualization. Figure 2 shows the performance comparison between four kernels for several NAS [1] Parallel Benchmark (NPB) codes, on a 16 processor cluster. CHAOS [2] is an HPC performance oriented kernel developed at Lawrence Livermore National Lab (LLNL). Xen kernel is a paravirtualized 2.6.12 kernel. The two RHEL kernels are off-the-shelf Red Hat Enterprise Linux version 2.6.9 and 2.9.12 respectively. In this figure, the y-axis shows the performance of different codes on the x-axis, relative to Livermore's CHAOS kernel. Using the averages illustrated here and several statistical techniques, we were able to empirically prove minimal performance degradation for paravirtualized systems. A comprehensive performance evaluation of paravirtualization in HPC should be found in [6].

During the execution of the application, the ASL is self evolving and dynamically adopting system which keeps modifying itself to meet the application's requirements at runtime and enhance its performance. Meanwhile, ASL provides a kernel adaptation system call that the user can interact with to suggest dynamic adaptation of the system during runtime. This dynamic customization constitute our final customization phase.

Much prior work in the area of application-specific operating systems (OSs) has thoroughly studied extensibility, specialization, and minimization of the OS in general. However, our research is novel in that it combines and extends these efforts into a system that automatically customizes the Linux operating system for a single appli-

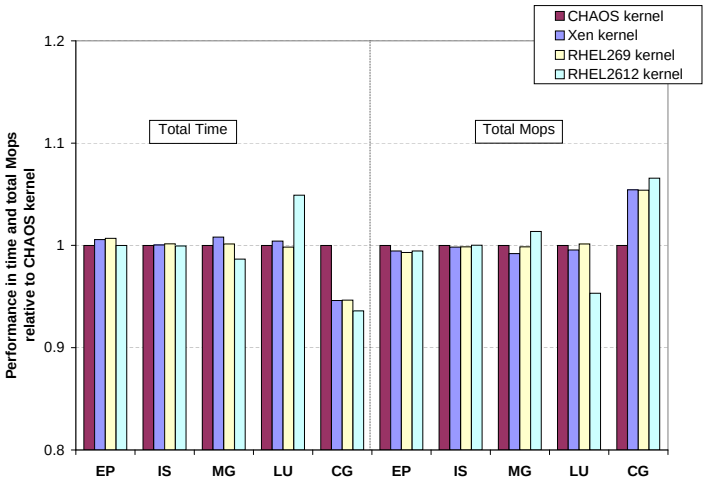


Figure 2: NAS Parallel Benchmark performance relative to CHAOS. The left half (first benchmark set) is for total time (lower is better); the right half is for Mops (higher is better).

cation run and is specifically focused on the application domain of scientific computing using high-performance clusters. At the same time, our use of Linux ensures that the environment for scientific applications developers remains familiar and unified across the development and production platforms they use; thereby promoting ease-of-use, programmer productivity, and efficient program management. Our Research's end-goal is a software system that automatically enables high performance scientific computing on commodity systems through application-specific customization and dynamic adaptation of a low-cost, popular, and familiar Linux operating system.

9

3. REFERENCES

[1] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. *The International Journal of Supercomputer Applications*, 1995.

[2] CHAOS Project. `http://www.cs.umd.edu/projects/hpsl/ResearchAreas/PerfPred.htm`.

[3] R. Gioiosa, F. Petrini, K. Davis, and F. Lebaillif-Delamare. Analysis of System Overhead on Parallel Computers. In *The 4th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT 2004)*, Roma, Italy, December 2004. Available from `http://www.c3.lanl.gov/~fabrizio/papers/isspit04.pdf`.

[4] C. Krintz and R. Wolski. Using Phase Behavior in Scientific Application to Guide Linux Operating System Customization. In *Workshop on Next Generation Software at IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2005.

[5] J. Marotzke and R. G. et al. Construction of the adjoint MIT ocean general circulation model and application to Atlantic heat transport sensitivity. *Journal of Geophysical Research*, 104(C12), 1999.

[6] L. Youseff, R. Wolski, B. Gorda, and C. Krintz. Paravirtualization for HPC Systems. In *Under submission to SuperComputing (SC 06)*, 2006.

[7] L. Youseff, R. Wolski, and C. Krintz. Linux Kernel Specialization for Scientific Application Performance. Technical Report UCSB Technical Report 2005-29, Univ. of California, Santa Barbara, Nov 2005.

Classification of Abnormal Activities in Video

Justin Muncaster
UCSB Computer Science Department
Four Eyes Lab
jmunk@cs.ucsb.edu

ABSTRACT
In multimedia computing the recognition of abnormal activities is becoming a major area of research interest. With applications in human-computer-interaction, elder care, security, and surveillance there is a strong push for advances in our ability to recognize both normal and abnormal activities at the semantic level. We use a probabilistic, hierarchical representation of activities to do recognition and provide an automatic way to define the low-level states. We classify abnormal activities meaningfully in terms of known high-level activities and show brief results of this work.

Categories and Subject Descriptors
I.2.6 [**Artificial Intelligence**]: Learning – *probabilistic modeling.*

General Terms
Algorithms, Design, Experimentation.

Keywords
Activity recognition, Bayesian networks, deterministic annealing.

1. INTRODUCTION
Activity recognition in video analytics has developed fast in recent years. There are a wide range of applications of activity recognition, including the monitoring of individuals within a shopping store, office buildings, hospitals, and banks. A variety of algorithms exist for processing of video sequences for key low-level functions, such as motion detection and object tracking, allowing one to identify simple activities such as walking or running.

Recently researchers have shown interest in recognizing complex behaviors in video at a higher semantic level (e.g., loitering, fighting). Such events typically require several sub events that occur in sequence before the complex event can be correctly decided. The decomposition of higher level events into sequences of lower level events suggests a hierarchical representation for events.

There are many papers that have tried to classify activities at the semantic level. In [7], Hongeng et al. recognizes multi-state activities using a hidden Markov model (HMM) and a particular form of static Bayesian network; [8] defines a series of rules, e.g. entry violation, escort, theft, possess, belong; Ryoo and Aggarwal [9] uses a context-free grammer based representation to represent composite actions and interactions. Duong et al. [10] use a switching duration hierarchical semi-Markov movel (S-HSMM) to model complex activities and they imposed Coxian distribution for duration model of the states.

In our previous work [1] we defined a general hierarchical framework in which to build models for activity recognition. We propose a probabilistic model that exhibits hierarchy and accounts for events of varying length. We also propose a way to automatically define low-level events. We review those results here and our method for recognizing abnormal activities within this framework. We then conclude with future directions.

2. MODELING ACTIVITIES

2.1 Representing hierarchy
The hidden Markov model (HMM) and its extension called the hierarchical hidden Markov model (HHMM) has been a powerful tool in speech recognition [2,3]. In a HMM one models the state of the system as a hidden random variable that probabilistically switches from one state to another. This random variable also probabilistically "emits" observations at each time slice. The learning problem is one in which we wish to discover the probabilities associated with this model using training data. The inference problem is to compute the probability distribution of the hidden variable given the test observations.

In a hierarchical hidden Markov model instead of emitting observations the HHMM emits another "sub-HMM", which can in turn emit further HMMs. The observations are then based on the states of each "level" and used to infer the state of the system. In [3] the author showed that this model was a special case of a dynamic Bayesian network (DBN) [3] and gave way how to model both the HMM and the HHMM in the DBN framework.

In [1] we proposed a "bare-bones" representation of a hierarchical system in the DBN framework, which omits the optional the dependencies given in [3]. Figure 1a shows a 3-level bare-bones hierarchical model. This isolates exactly what makes the model hierarchical and casts all other dependencies as domain-specific. This reduces the number of model parameters which is beneficial for both learning and inference.

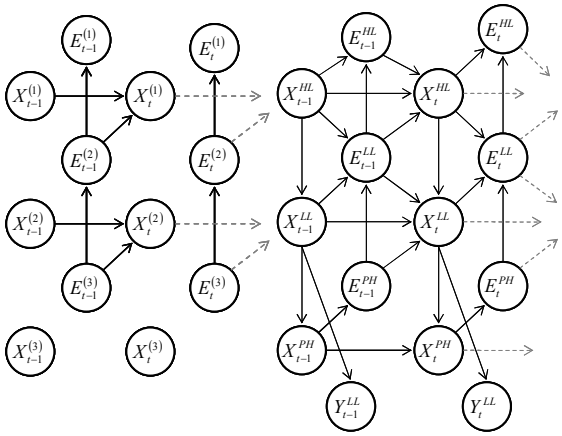


Figure 1 (a) The bare-bones HDBN as we defined it (b) The actual model used in our experiments.

2.2 Defining low-level states

To define low-level states we take our observations and suppose that they are generated from the low level activity state. We then perform clustering using the deterministic annealing algorithm [5], partitioning the feature space into discrete blocks. Each cluster center corresponds to a low-level state of the model.

After we perform clustering we then fit a Gaussian to each cluster center, giving us the probability density for our low-level observations $\mathbf{y}_t^{LL}$,

$$\Pr\left(Y_t^{LL} = \mathbf{y}_t^{LL} \mid X_t^{LL} = k\right) = N\left(\mathbf{y}_t^{LL}; \boldsymbol{\mu}_k^{LL}, \boldsymbol{\Sigma}_k^{LL}\right),$$

where $\boldsymbol{\mu}_k^{LL}$ is a mean vector and $\boldsymbol{\Sigma}_k^{LL}$ is a covariance matrix.

3. EXPERIMENT

3.1 Data set

We tested our algorithm using the video clips of a shopping center in Portugal that we found in [6]. We identify three high level activities: Entering the shop, leaving the shop, and passing the shop. For each video we also ran a tracker developed by [4] to track people in the image plane. We randomly set aside 8 tracks for training (3 entering, 3 leaving, 2 passing) and 6 tracks for testing. We labeled each frame in the training data with the appropriate high-level event.

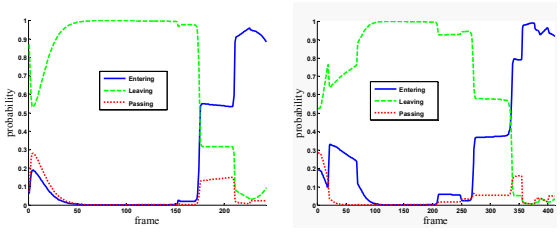
We used a four-dimensional feature space: x-position, y-position, x-velocity, y-velocity. The velocity estimate was done using fixed-lag differences of the positions and all features were smoothed with a Gaussian kernel. The training data points were then used to compute the whitening matrix, which we used to normalize all feature points.

We set up our model structure as shown in Figure 1b. In this model X_t^{HL} represents the high level activity at time *t*, and E_t^{HL} denotes whether the high level sequence has ended at time *t*. There are analogous meanings for the low-level state (LL) and the phase distribution (PH) which models the duration of the low-level activity.

3.2 Results

Our results for normal activities are given in [1] and are promising. In Figure 2 we show results for some abnormal activities. In these activities we use data that is unlike any data that we trained on.

In Figure 2 we have two video sequences with tracks that we could not label with one of our high level events. These videos both had the following behavior: A person would exit the store, walking out and turning the corner. Next, the person would stop, pause for a short time, and turn around and go right back into the store. The trajectory of the person is unlike any trajectory seen in our training data. However, as we can see in Figure 2, we are able to infer a meaningful label for both tracks where this occurs.



(a) (b)

Figure 2. Results for leaving and reentering.

The results of figure 7 also suggest that our duration model is working effectively. Notice that the track in Figure 7a is shorter (i.e., has fewer frames) than the track in Figure 7b. In fact, each low-level event also occurred for a shorter period of time. We believe that because the uncertainty associated with the duration of a low level event is effectively modeled, we are able to handle this case where events occur at different speeds.

4. CONCLUSION

In this work we have given a short synopsis of work done to robustly classify abnormal activities in video. In future work we wish to address the question regarding how to classify abnormal sequences that bear little or no resemblance to any of our predefined high level events. We are working on ways to augment our model to be able to recognize such abnormal activities automatically without having to rely on labeled data of abnormal activities.

5. REFERENCES

[1] J. Muncaster and Y. Ma, Activity recognition using dynamic Bayesian networks with automatic state selection, *To be submitted*.

[2] S. Fine, Y. Singer, and N. Tishby. The hierarchical Hidden Markov Model: Analysis and applications. *Machine Learning*, 32:41, 1998.

[3] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference, and Learning*. Ph.D. thesis, UC Berkeley, 2002.

[4] Y. Ma and Q. Yu, Multiple hypothesis target tracking using merge and split of graph's nodes, ISVC 2006.

[5] K. Rose. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proceedings of the IEEE*, 1998.

[6] http://groups.inf.ed.ac.uk/vision/CAVIAR/CAVIARDATA1

[7] S. Hongeng, F. Bremond and R. Nevatia, "Bayesian Framework for Video Surveillance Application", ICPR 2000, Vol I, pp. 164-170, September 2000.

[8] V. D. Shet et al., "Multivalued Default Logic for Identity Maintenance in Visual Surveillance" ECCV 2006.

[9] M. Ryoo and J. Aggarwal, Recognition of Composite Human Activities through Context-Free Grammar Based Representation, 1709- 1718, CVPR 2006.

[10] Duong et al., Activity recognition and abnormality detection with the switching hidden semi-markov model, CVPR 2005

Application Testing and Analysis for Security

Greg Banks, Marco Cova, Viktoria Felmetsger, Richard Kemmerer, Giovanni Vigna
Reliable Software Group
{nomed,marco,rusvika,kemm,vigna}@cs.ucsb.edu

1. INTRODUCTION

In an ideal world, programs would be developed by experienced programmers who possess solid security skills and who follow sound methodologies. Unfortunately, in reality, applications are designed, developed, and deployed under strict time constraints and with little emphasis on security. As a consequence, vulnerabilities — exploitable locally, remotely, or both — are discovered and publicly disclosed on a daily basis. The impact of these vulnerabilities can be severe, and the cost of correcting errors after an application has been deployed can be very high.

To ameliorate these security problems, it is necessary to develop tools and techniques to improve the security of applications. The most effective approach would be to provide mechanisms that can improve the software development cycle in order to help all developers write solid security-aware code. Unfortunately, this is not always possible. Instead, a second line of defense, namely testing and auditing application code for possible security problems, is frequently used.

In this paper we introduce two tools that we have developed to identify security-critical vulnerabilities in applications for which source code might not be available. In Section 2 we introduce an automated testing technique, called *fuzzing*, and present SNOOZE, a tool for building flexible, security-oriented, network protocol fuzzers. In Section 3 we discuss a tool that we implemented for static detection of vulnerabilities in x86 executables. Finally, in Section 4 we discuss some benefits and drawbacks of both approaches and highlight directions for future work.

2. SNOOZE

Fuzzing is a well-known black-box approach to the security testing of applications. Its basic idea is to provide a system with unexpected, random, or faulty input with the purpose of exposing corner cases that were not considered during implementation [3]. This technique has several benefits: first, it can be applied to programs whose source code is not available; second, it can be used on large applications; third, bugs found with fuzzing are reachable through user input, and, as a consequence, are exploitable.

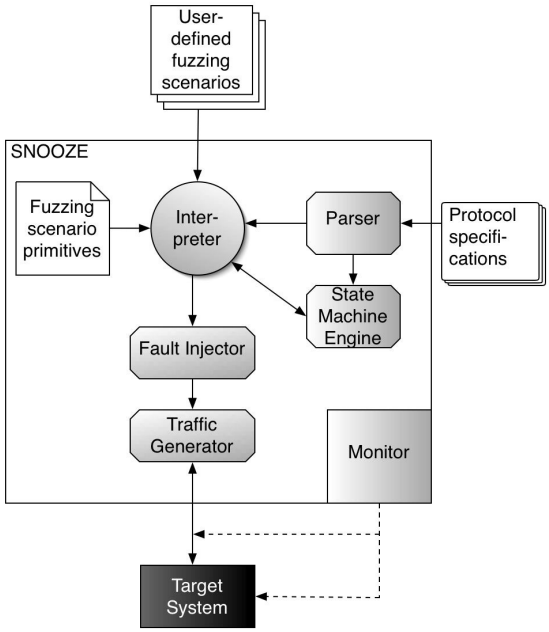


Figure 1: Main components of SNOOZE.

As a result, a number of tools have taken this approach to testing systems. However, in general, the ability to generate input in a manner that is both unexpected or faulty *and* likely to trigger a well-known, target-specific, attack (e.g., SQL injection) is lacking. Furthermore, support for testing complex applications that implement stateful protocols is generally absent. As a result, in [1] we proposed and implemented a prototype of SNOOZE, an extensible tool for development of network protocol fuzzers.

Figure 1 shows the high-level architecture of SNOOZE. The *Interpreter*, the component responsible for running the tests, takes as input a set of *protocol specifications*, a set of user-

defined *fuzzing scenarios*, and a module implementing *scenario primitives* (i.e., operations to send or receive protocol messages and navigate the state machine). A protocol specification defines the general characteristics of a protocol, such as the format of header fields and their default values, the syntax of messages that can be exchanged in the protocol, and the allowed message flows (i.e., a state machine). The default values assigned to fields can be changed and additional fields can be inserted through the use of the mutation primitives provided by the *Fault Injector* component.

To fuzz an application a fuzzing scenario must be generated. This scenario makes use of the primitives and protocol specifications already present in the tool. It simply encodes the message sequence that it wants to be played with the server as well as the type of fuzzing to be done on the messages in this sequence.

We used SNOOZE to test several implementations of SIP, a widely-used signaling protocol for VoIP applications, and found several previously unknown vulnerabilities in an application making use of one of these implementations.

3. A BINARY ANALYZER

In several cases, programs are distributed in binary-only format, e.g., to protect proprietary intellectual property. The executable code of an application, however, often contains enough information to determine whether the application contains vulnerabilities.

We explored this idea by building a tool that detects "tainted-data-to-system" vulnerabilities in x86 executables. `system()` is a function provided by the standard C library that spawns a shell to evaluate its parameter and execute the associated command(s). If the parameter content is under user control, it could be used by an attacker to execute arbitrary commands under the privileges of the vulnerable application. Thus, these vulnerabilities are especially critical in SUID binaries and remotely-accessible applications.

We detect such vulnerabilities using binary analysis [5] and static analysis techniques [4]. Binary analysis extracts useful information from executable code. Static analysis predicts safe and computable approximations of the behaviors that the application could show at run-time. Using binary analysis, we perform several tasks: we disassemble an executable, reconstruct its control flow graph (CFG), extract information regarding the library functions that it invokes, identify functions through which untrusted (*tainted*) data is read into the program and sensitive functions (e.g., `system()`) that should only use trusted data. Using static analysis, we detect whether it is possible for tainted data to reach such sensitive functions. More precisely, we use symbolic execution [2] to simulate execution along all feasible paths of the application using symbolic, rather than concrete, inputs. This way, we compute the approximate behavior of the application with all possible input values.

We used our tool to detect the use of tainted data in calls to the `system()` function in several typical Linux utilities.

4. CONCLUSIONS

In this paper, we presented our experience with the problem of automatically identifying security-relevant bugs in applications. We presented two orthogonal approaches to the solution of this problem and two tools that implement them.

We discussed SNOOZE, a fuzzing-based tool for the testing of network applications. SNOOZE employs a black-box, dynamic approach, i.e., it does not assume any *a priori* knowledge of the implementation details of an application and works by testing a running instance of that application. Unlike existing fuzzers, it also provides methods to test applications that implement complex, stateful protocols. The advantages of this approach are that it is relatively simple to understand and implement, yields no false positives, and allows a tester to quickly find "shallow" bugs. However, similar to other testing methodologies, it cannot provide any guarantee of finding all bugs existing in an application.

Second, we presented a tool that detects taint-style vulnerabilities in x86 executables. It uses a white-box approach, thus requiring the binary image of an application to be available; it is static in that it does not require one to actually run the application under test; finally, it is based on tracking the flow of tainted data through a program. This approach performs a more thorough analysis of an application and, therefore, is capable of detecting "deep" bugs that are difficult to uncover through testing. However, it requires a non-trivial infrastructure to be built. In addition, it is dependent on the specific implementation details of an application and requires the use of several heuristics to make the problem tractable.

In the future, we intend to explore how static and dynamic approaches can be combined to maximize each other's strengths and minimize limitations. Currently, we are also investigating how other analysis techniques, such as model checking, can be applied to finding security problems in applications.

5. REFERENCES

[1] G. Banks, M. Cova, V. Felmetsger, K. Almeroth, R. Kemmerer, and G. Vigna. Snooze: toward a stateful network protocol fuzzer. In *In Proceedings of the 9th Information Security Conference*, 2006.

[2] J. C. King. Symbolic Execution and Program Testing. *Communications of the ACM*, 19(7):385–394, 1976.

[3] B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of unix utilities. *Communications of the ACM*, 33(12):32–44, 1990.

[4] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[5] T. Reps, G. Balakrishnan, J. Lim, and T. Teitelbaum. Next-Generation Platform for Analyzing Executables. In *Proc. of the 3rd Asian Symposium on Programming Languages and Systems*, 2005.

Simulation of Large Scale Sensor Network

Ye Wen
University of California, Santa Barbara
wenye@cs.ucsb.edu

Rich Wolski
University of California, Santa Barbara
rich@cs.ucsb.edu

ABSTRACT

Simulation is an effective tool for developing, evaluating and analyzing sensor network applications, especially when deploying a large scale sensor network remains an expensive and labor intensive endeavor. In this short paper, we describe our current work on the simulation of large scale sensor networks. More specifically, we introduce the DiSenS (DIstributed SENsor network Simulation) system – a distributed software infrastructure for scalable sensor network simulation. We also discuss how different sensor devices can be coordinately simulated in the infrastructure to support the simulation of heterogeneous sensor network. At the end, we briefly discuss some on-going and possible future work of our project.

1. INTRODUCTION

Sensor networks make possible the instrumentation and actuation of potentially a large variety of environmental phenomena. By making the provisioning of computing power non-invasive and inexpensive, the ability to apply computation as a way of analyzing "the world" ubiquitously becomes a possibility, the potential impact of which cannot be overstated.

However, despite the potential for transformative scientific and even social change that sensor networks seem to promise, their development is, at present, still nascent. While various technological and economic obstacles exist, a key impediment to their development is the lack of a scalable simulation capability that provides the fidelity necessary to support both coherent design and efficient engineering of sensor network systems. State-of-the-art sensor network research and development relies on labor and resource intensive trial-and-error using physical devices and *in situ* deployments. Few other systems of similar complexity and potential expense (e.g. computational processors, embedded systems, network architectures, etc.) are investigated and engineered in the same way: without high-quality and multi-fidelity simulation support.

To accelerate possible research and development advances for sensor networks, our work focuses on the development of a sensor network simulation capability in the form of DiSenS – a distributed software infrastructure for scalable sensor network simulation. DiSenS is intended to serve as a research tool for the development of simulation models targeting different fidelity levels, and to allow these investigations to take place at scales unattained by previous systems.

Previous works including discrete-event sensor network simulation systems and full-system sensor network simulators have made big effort to address the similar problem. However, none of them is without limitations in achieving both scalability and fidelity. Discrete-event systems such as those described in [2, 4, 8] model device functionality and communication as a set of partially ordered events modifying distributed state. Often, these systems have focused on communication interactions (which takes place via unreliable and difficult-to-model communication radios) and only roughly approximate the behavior of the constituent devices themselves. By sacrificing device fidelity, discrete event simulators can achieve very high performance and scale well. Full-system simulators [6, 10, 5] take an alternative approach. They simulate the internal device functionality in detail and allow ensemble behavior to emerge from the interactions of independent-but-communicating simulated devices. These systems achieve sufficient fidelity levels, but the need to coordinate multiple simulated devices has limited their scalability.

Our work attempts to extract and combine the benefits of both approaches. DiSenS supports high-fidelity and high-performance emulation of individual sensor devices as well as radio communication simulation that scales. The infrastructure is designed to use dedicated clusters of commodity "PC" class machines interconnected by high-performance local-area networking technology (heretofore termed "cluster computing" technology).

Our work also tries to extend the simulation system to support heterogeneous devices. At the current stage, DiSenS is able to simulate the basic sensor device, i.e. motes (including both Mica2 and MicaZ) and the intermediate sensor device, Stargate, with cycle-closeness. These different devices are simulated coordinately in accordance to their relative speed to achieve the overall fidelity and performance of a heterogeneous sensor network.

2. SCALABLE SIMULATION OF LARGE SCALE SENSOR NETWORK

At the core of DiSenS is a hardware emulator with extensive support for various popular sensor network devices. In the current implementation, we emulate the mote [3] devices (the Mica2 and MicaZ platforms), Stargate devices [7], and iPAQ devices [1] and we are adding the support of other devices, like Telos [9]. Thus, the system is capable of heterogeneous sensor network simulations. As

the basis for accurate simulations, the hardware emulator provides cycle-accuracy (for motes) or cycle-closeness (for Stargate) simulation.

The hardware emulator is able to be parameterized by a set of pluggable fidelity enhancing models, e.g. radio model, power model, etc., to allow experimentation with different fidelity levels and modes of investigation.

Given the fidelity provided by the full-system simulation functionality, our primary goal is then to scalablly simulate "large" ensembles of sensor nodes so that potential problems of scale can be studied. The way we approach this problem is to use computer cluster to distributedly simulate a large set of emulated device instances and the radio communication among them. The challenge is then how to efficiently coordinate the simulation progress for each emulated device in a distributed memory system with large message passing latency. DiSenS uses a peer-to-peer simulation design: each device is emulated in a separated thread and keeps its own clock. The individual node simulation threads are then glued together by a simple and efficient synchronization protocol, which makes the complete simulation scalable to a large size of distributed computation resources. To further efficiently use the computing resources, we also apply the state-to-art graph partition algorithm to distribute the sensor nodes among computation hosts intelligently to achieve the optimal performance.

As the result, using commodity cluster hardware, DiSenS can simulate one node approximately **9** times faster than real time speed, **160** nodes in real time speed using 16 dual-processor machines and **8192** nodes at nearly tenth of real time speed, which is **32** times of that reported previously [2]. Figure 1 shows the scalability of the simulation of sensor networks for both 1-D and 2-D topologies on a 16-node cluster in our lab.

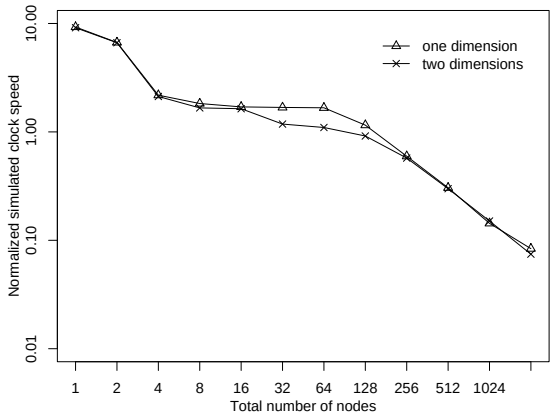


Figure 1: Best performance comparison of 1-D and 2-D topology on a 16-node cluster. X-axis is total number of nodes simulated. The Y-axis is normalized performance.

3. FUTURE WORK

The initial work on DiSenS lays the foundation for a lot of exciting possible extensions and applications. One of our current on-going works is to study the statistical properties of radio transmission of sensor networks and incorporate them into the simulation framework, so that efficient and accurate network simulation is possible. We are also trying to build a sophiscated sensor network application development environment upon DiSenS to utilize its fidelity, scalability and flexibility.

As the future work, we want to extend the distributed simulation framework to support the simulation in a heterogeneous computation environment, like Grid environment, to explore the possibility of massive sensor network simulation. We are also thinking about coupling the simulation with actual hardware deployment, so that virtuality and reality can enhance each other.

4. REFERENCES

[1] iPAQ devices. `http://welcome.hp.com/country/us/en/prodserv/handheld.html`.

[2] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. *ACM Conference on Embedded Networked Sensor Systems*, Nov. 2003.

[3] Mote hardware platform. `http://www.tinyos.net/scoop/special/hardware`.

[4] S. Park, A. Savvides, and M. B. Srivastava. SensorSim: a simulation framework for sensor networks. *ACM International workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 104–111, 2000.

[5] J. Polley, D. Blazakis, J. McGee, D. Rusk, and J. S. Baras. ATEMU: A Fine-grained Sensor Network Simulator. *IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, 2004.

[6] Simulavr: A simulator for the Amtel AVR processor family. `http://www.nongnu.org/simulavr`.

[7] Stargate: a platform X project. `http://platformx.sourceforge.net/`.

[8] S. Sundresh, W. Kim, and G. Agha. SENS: A Sensor, Environment and Network Simulator. *The IEEE 37th Annual Simulation Symposium*, 2004.

[9] Moteiv Corporation. Telos Sensor Network Module. `http://www.moteiv.com/`.

[10] B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora: Scalable Sensor Network Simulation with Precise Timing. *The Fourth International Symposium on Information Processing in Sensor Networks*, Apr. 2005.

A Sketch Interface to Aid 3D Scene Reconstruction

Jonathan Ventura
Four Eyes Lab
jventura@cs.ucsb.edu

1. INTRODUCTION

The main challenge of scene reconstruction is to synthesize new views of a scene using photographs as input. Previously researched methods require varying levels of user input.

The "Tour into the Picture" (TIP) system allows a user to work in the two-dimensional space of a single photograph to achieve a three-dimensional reconstruction [5]. This reconstruction models the scene as a box, much like a diorama. The user places the vanishing point and an "inner rectangle" indicating the back wall of the box, along with rectangles indicating foreground objects. To reconstruct the scene, the photograph is projected onto the box and foreground objects. This system has several disadvantages. The correct placement of the vanishing point is not always obvious in photographs without a visible horizon. In addition, the system requires an alpha mask for the foreground objects, and a separate retouched photograph in which the foreground objects have been removed, to fill in the pixels behind popped-up foreground objects. Thus the system is not completely contained in the user interface.

A later system, "Automatic Photo Pop-Up," presents a first attempt at a fully automatic reconstruction system, using a TIP-style approach [4]. The system must be trained on a collection of hand-segmented images, and in practice has been shown to correctly segment and reconstruct only about 30% of input images. More work is needed before a fully-automatic system can be robust enough to be practical for a wide variety of typical images.

A compromise approach is needed which incorporates user input intelligently by leveraging information in the photograph. Our goal is a fast system which is easy to use and learn. To this end, we propose a semi-automatic scene reconstruction method for a single photograph which uses a gestural interface as user input. This gestural system uses image features in several ways in order to lessen the the cognitive load on the user. Because strokes on the image form the only input to the system, it is easily adaptable to a tablet display with a pen device for input.

2. PROPOSED METHOD

We propose a simplified scene model suggested by [6] which is more general than that of TIP or Automatic Photo Pop-Up. This scene model assumes that the photograph consists of (i) a ground plane which meets the sky at the horizon and (ii) foreground objects in the form of planes or boxes protruding from that ground plane.

2.1 Vanishing Line

The first step in fitting the scene model to an image is determination of the ground plane. In this system, we find the ground plane by the horizon, or vanishing line.

As illustrated in figure 1, we assume that the top corners of the image (points 1 and 2) and the intersection points of the vanishing line with the image boundaries (points 3 and 4) are far away, and that the bottom two corners of the image (points 5 and 6) are points on the ground plane intersecting with the image plane. Points 1 - 4 are ideal points of their respective projected points on the image plane, and points 5 - 6 intersect the image plane. The equations to find the coordinates of the points and the parameters of the ground plane are given in [6].

The simplest method of specifying the vanishing line is for the user to explicitly draw the vanishing line with one stroke. However, if a horizon line is not clearly visible in the image, the correct placement of the vanishing line may not be obvious to the user. In these cases the user may prefer to specify a rectangle on the ground plane. The system will then extend the sides of this rectangle to find their vanishing points, and connect these points to determine the vanishing line. Alternatively, the user may specify at least two pairs of parallel lines, which can be similarly extended to determine the vanishing line.

2.2 Foreground Objects

Once we have determined the ground plane, we would like to "pop up" foreground objects such as walls and people. Two to three strokes are needed for this gesture. The first stroke is along the ground plane, indicating the width of the object on the ground. The second stroke is up along the side of the object, indicating the height of the object off of the ground. If desired, a third stroke along the ground plane can be supplied to indicate the object's depth. Without this

third stroke, the object is assumed to be planar (having no depth).

Figure 2 shows a sample scene with two background walls, two foreground objects, and no clear horizon. Strokes have been placed on the image to indicate typical gestures that would be used to mark up the scene. To specify a stroke, the user clicks and drags the mouse or pen from one endpoint of the line to the other. First, a rectangle has been traced on the hardwood floor, allowing the system to find the vanishing line. Next, the background walls have been traced using two strokes each, one on the floor, and one along the corner of the room. The lamp is modeled as a planar object by using two strokes to give width and height. The chair is modeled as a box by giving three strokes for width, height, and depth.

3. IMPLEMENTATION AND EVALUATION

The system is implemented in C++ using OpenGL. Two modes are available to the user: an editing mode, where strokes on the image are translated into gestures; and a reconstruction mode, where a synthesized view of the scene can be repositioned in real-time.

3.1 Gestures

A single stroke consists of clicking, dragging, and releasing the mouse or pen device. Several strokes in succession constitute a gesture. When the first stroke is received, the system starts a new gesture. Each following stroke is added to that gesture. If, after, a certain amount of time (in our system, two seconds), no more strokes are received, the gesture is considered finished. At that time the system attempts to parse the gesture.

If the gesture has only one stroke, it is assumed to be indicating the vanishing line, so the system extends the line to the borders of the image and recalculates the ground plane. If the gesture has four strokes, then the system assumes that two pairs of parallel lines on the floor are being indicated. In this case the lines are extended to find vanishing points and from there the vanishing line is updated.

A gesture with two strokes is assumed to be indicating a planar object. The first stroke on the ground plane determines the plane of the object. To find this, we cast rays from the origin through the image plane at the stroke endpoints to the ground plane. The two intersection points on the ground plane along with the ground plane normal determine the object plane. Then we cast rays through the endpoints of the second stroke to the object plane to determine the height of the object.

A gesture with three strokes indicates a box object. The system follows the same behavior as for two strokes, and then uses the third stroke to determine object depth. This is done by finding the location of the third stroke on the ground plane (using ray casting), and extruding the object plane along that stroke.

3.2 Conclusions and Future Work

The goal of our system is a fast and intuitive interface for scene reconstruction on a single photograph, suitable for a tablet display. To this end we avoided any use of the keyboard or multiple mouse buttons for input. We also wanted to avoid traditional GUI elements such as windows, menus, or dialog boxes, and tried to only display the image and visual feedback on top of it.

Some properties of our design hinder its effectiveness as a fast and intuitive interface. The user must allow time between gestures, because the system has no indication for the end of a gesture other than a time delay. This could be improved by the introduction of a stroke or set of strokes to indicate the end of a gesture. Also, because the system differentiates gestures only by their number of strokes, we have not allowed for the addition of new gestures which use the same number of strokes as one of the existing gestures. To this end we would like a more sophisticated way of differentiating gestures which looks at the nature of the stroke.

A second goal of our system is that we wanted to use image analysis as a second form of input, in order to reduce the amount of user input needed and improve the reconstruction quality (see figure 3). This goal is not met by our current implementation; however, we have two ideas we would like to pursue in this direction. The first is to incorporate image segmentation [7] and region filling [1] techniques which can automatically segment and remove a foreground object and fill in the missing background behind it, eliminating the need for the alpha mask and retouched image used in TIP (see figure 4). The second is to incorporate edge-snapping techniques [3]. This would allow for strokes to be aligned with lines in the image, which would make some gestures (such as the rectangle on the floor) easier and more accurate.

4. REFERENCES

[1] Criminisi, A., Pérez, P., and Toyama, K. Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on Image Processing 13*, 9 (September 2004).

[2] Debevec, P. E., Taylor, C. J., and Malik, J. Modeling and rendering architecture from photographs. In *SIGGRAPH* (1996).

[3] Gleicher, M. Image snapping. In *SIGGRAPH* (1995).

[4] Hoiem, D., Efros, A. A., and Hebert, M. Automatic photo pop-up. In *SIGGRAPH* (2005).

[5] Horry, Y., Anjyo, K.-I., and Arai, K. Tour into the picture: using a spidery mesh interface to make animation from a single image. In *SIGGRAPH* (1997).

[6] Kang, H. W., and Shin, S. Y. Extended rendering: Tour into the picture using a vanishing line and its extension to panoramic images. In *Computer Graphics Forum* (2001), vol. 20.

[7] Rother, C., Kolmogorov, V., and Blake, A. Grabcut: Interactive foreground extraction using iterated graph cuts. In *ACM Trans. on Graphics (SIGGRAPH'04)* (2004).

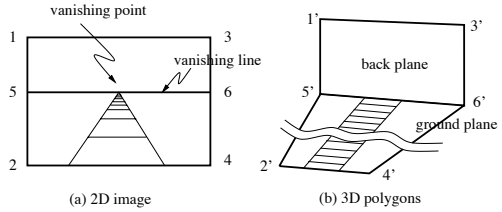


Figure 1: The vanishing line in image space and reconstruction space. Figure from Kang, et al.

Figure 2: A sample image with gestures for vanishing line, two walls, and two foreground objects.

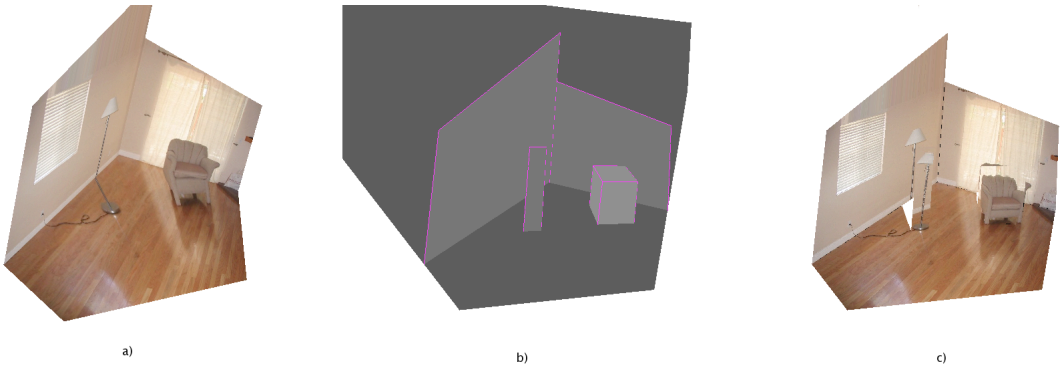


Figure 3: Three synthesized views of the scene from figure 2: a) The reconstruction without the foreground objects. b) The primitives used to model the scene. c) The reconstruction with foreground objects. Image segmentation and inpainting techniques could be used to remove artifacts.

Figure 4: An image generated using a hand-made foreground mask and retouched background image. We would like to be able to produce these supplemental images semi-automatically.

Analyzing the Phase Behavior of the Circadian Clock in *Arabidopsis thaliana*

Stephanie R. Taylor
Computational Science and Engineering Lab/Doyle Group
Department of Computer Science
staylor@cs.ucsb.edu

Francis J. Doyle III
Doyle Group
Department of Chemical Engineering
frank.doyle@icb.ucsb.edu

Linda R. Petzold
Computational Science and Engineering Lab
Department of Computer Science
petzold@cs.ucsb.edu

ABSTRACT
In this paper, we utilize the impulse phase response curve as a tool to investigate the timing behavior of mathematical models of oscillatory systems. We analyze two models of a plant circadian clock and demonstrate that the key target for improvement is the mechanism for light perception.

Categories and Subject Descriptors
J.3 [**Life and Medical Sciences**]: *biology and genetics*

General Terms
Algorithms, Measurement, Experimentation

Keywords
systems biology, gene regulatory network, sensitivity analysis, limit cycle oscillator, circadian clock, *Arabidopsis thaliana*

1. INTRODUCTION
Animals and plants follow a nearly 24-hour cycle of behavior that is controlled by an internal clock which, in turn, is entrained to its environment [2]. The clock mechanisms for animals such as fruit flies and rodents have been studied extensively and are reasonably well-understood. Biological experimentation and mathematical models have been used to elucidate many of the mechanisms involved in the regulation of their circadian rhythm. Study of the plant clock is at a much earlier stage. Although evolution has conserved some aspects of the clock across kingdoms - all have interlocked feedback loops (including negative feedback), we cannot simply co-opt the models for animals and search for analogous genes/proteins in plants. One aspect that is not common across kingdoms is the mammalian master clock, which resides in the hypothalamus and regulates the clocks in other organs. Because plants have no such master clock, and because they are rooted to the ground and cannot flee from unfriendly environments, the individual clocks in cells must be robust to changes in environmental perturbations such as light intensity and temperature change [4].

2. MATHEMATICAL MODELS
Model development is part of an iterative process; as new clock components are identified and incorporated into our conceptual model, they must, in turn, be incorporated into the mathematical model. Then, through model simulation, we can assess our conceptual understanding of the mechanisms involved. We use sensitivity analysis to determine the robustness of different model architectures and to streamline these model-development iterations.

Recent work has produced a sequence of candidate mathematical models [6, 8] of the circadian clock in the plant *Arabidopsis thaliana*. Like other clock models, each candidate *i)* consists of a set of ordinary differential equations that show stable oscillations in constant conditions, and *ii)* has incorporated into it a mechanism for light to enter and entrain the system to its environment. The system is represented mathematically as

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{p})$$

where $\mathbf{x}$ is the vector of states, and $\mathbf{p}$ is the vector of parameters.

The present work analyzes the most sophisticated of the models [8] (see Figure 1); they correctly predict the results of several gene knock-out experiments [3].

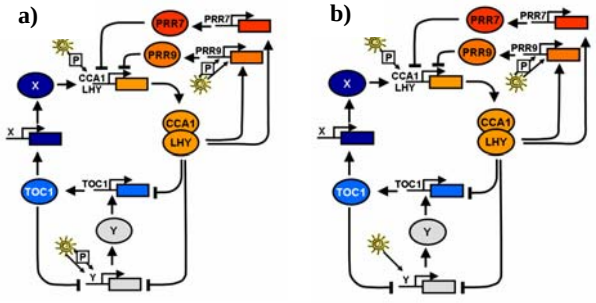


Figure 1. Schematics of the modeled gene regulatory network. a) Model A. Light enters the system through multiple pathways. In this model, light affects the transcription of gene Y both as a pulse at dawn and as a continuous stream. b) Model B. The layout is identical, with the exception that there is no pulsatile affect of light on the system. The two models share identical ordinary differential equations (with the noted exception), but have differing parameter sets.

3. PHASE RESPONSE CURVES
Sensitivity analysis quantifies the change in behavior of a system in response to a disturbance. Classical sensitivity analysis computes the overall impact of a parametric perturbation on the system. Oscillatory systems have characteristics, such as period, phase, and morphology (the shape of the limit cycle), which may be sensitive to parametric perturbations. For plant circadian clocks, the timing of physiological events (such as flowering),

must coincide with the appropriate time of day. The plant must be able to adjust its behavior to environmental cues such as changing seasons (and lengthening of daylight) - we say the phase of the plant behavior must coincide with the phase of the day. Since the timing of events is of such importance, in order to fully understand the mechanism of the clock we must understand how its sub-networks influence such timing. To do that, we utilize the impulse phase response curve (IPRC) we formulated in [7, 8]. Typically, PRC's are computed experimentally by applying a pulse of light to a free-running oscillator, which is then allowed to settle. Once the system has settled, we measure its phase shift in reference to an unperturbed system. Using the isochron-based phase sensitivity introduced by Kramer *et al.* [5], we determine the phase response curves for all parameters and states in a mathematical model. This is inexpensive to compute, requiring only one forward solve (of the system and its sensitivity equations) and one backward solve (of the adjoint equation).

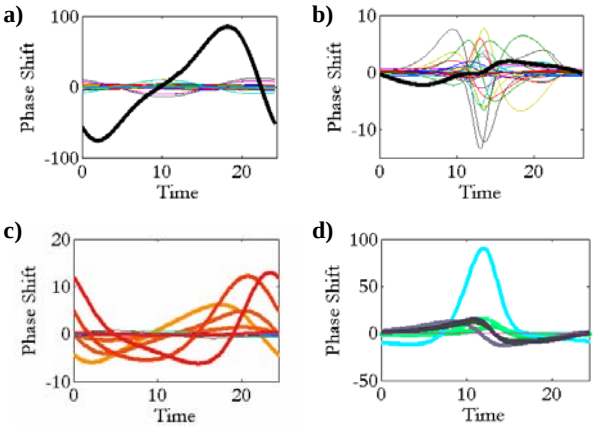


Figure 2. IPRC's. a) Parametric IPRC for Model A. The thick line represents the timing effects of light - an effect dwarfing that of all others. b) Parametric IPRC for Model B. The thick line represents the timing effects of light, which is much less severe. c) State IPRC for Model A. The thick lines represent LHY/CCA1 and PRR7 mRNA and protein – the negative feedback loop between these entities has the most control of the timing. d) State IPRC for Model B. The thick lines represent TOC1 and Y mRNA and protein. Their large magnitude indicates that negative feedback loop between TOC1 and Y completely dominates the timing.

4. ANALYSIS

For each model, we compute the parametric IPRC and the state IPRC (see Figure 2). When examining light, we expect it *i)* to have the strongest effect upon the system, but that it *ii)* will not completely dominate the system. For Model A, we find criterion *ii)* is violated because the system timing is ultra-sensitive to light. We perform a numerical experiment, computing the traditional PRC, and find that the system is reset to dawn whenever it encounters a pulse of light (data not shown). This verifies our insight that light has too strong an effect on the system. For Model B, we observe that light no longer has the strongest influence, thereby violating criterion *i)*. To investigate the mechanisms of the autonomous oscillator, we turn to the state IPRC's. For Model A, the states with the greatest effect upon the timing are those associated with LHY/CCA1 and PRR7, indicating the negative feedback loop involving them is the core mechanism behind the oscillations. For Model B, we see a dramatic difference - TOC1 and Y dominate the timing. In fact, Model B shows stable oscillations when LHY/CCA1 is removed (data not shown). Since biological experiments indicate that LHY/CCA1 is imperative for oscillations [1], intuition leads us to conclude that Model B's lack of dependence on LHY/CCA1 is unrealistic.

The high sensitivity of Model A to light and the relatively low sensitivity of Model B to light indicate that our understanding of the system's light perceiving mechanisms is inadequate. In addition, the behavior of LHY/CCA1 is affected by light [1], and Model B misrepresents LHY/CCA1. This strengthens the argument that the next generation model must include more detailed information about light perception.

5. ACKNOWLEDGMENTS

Melanie Zeilinger and Eva Farre played instrumental roles. This work was supported by the Institute for Collaborative Biotechnologies through grant DAAD19-03-D-0004 from the U.S. Army Research Office (FJD) and IGERT NSF grant DGE02-21715 (LRP).

6. REFERENCES

[1] D. Alabadi, M.J. Yanovsky, P. Mas, S.L. Harmer, and S.A. Kay. Critical role for CCA1 and LHY in maintaining circadian rhythmicity in Arabidopsis. *Curr Biol*, 12(9);757-761, Apr 2002.

[2] J.C. Dunlap, J.J. Loros, and P.J. DeCoursey, editors. *Chronobiology: Biological Timekeeping*. Sinauer Associates, Inc. Publishers, Sunderland, MA, USA, 2004.

[3] E.M. Farre, S.L. Harmer, F.G. Harmon, M.J. Yanovsky, and S.A. Kay. Overlapping and distinct roles of PRR7 and PRR9 in the Arabidopsis circadian clock. *Curr Biol*, 15(1):47–54, Jan 2005.

[4] A.J.W. Hall and H.G. McWatters, editors. *Endogenous Plant Rhythms*. Blackwell Publishing Ltd, 2005.

[5] M.A. Kramer, H. Rabitz, and J. Calo. Sensitivity analysis of oscillatory systems. *Appl. Math. Mod.*, 8:328–340, 1984.

[6] J.C. Locke, M.M. Southern, L. Kozma-Bognar, V. Hibberd, P.E. Brown, M.S. Turner, and A.J. Millar. Extension of a genetic network model by iterative experimentation and mathematical analysis. *Mol. Systems Biol*, 1, (2005).

[7] Taylor, S.R., Doyle III, F.J., Petzold, L.R., Phased and confused? A primer on isochrons, phase sensitivity, and PRC's. in preparation.

[8] M.N. Zeilinger, E.M. Farre, S.R. Taylor, S.A. Kay, and F.J. Doyle III. A novel computational model of the circadian clock in Arabidopsis that incorporates PRR7 and PRR9. *Mol. Systems Biol.*, accepted.