Software Engineering Institute
Carnegie Mellon

Research, Technology, and System Solutions Program

Effectively Serving the Changing Needs of DoD Missions

Research, Technology, and System Solutions Program

Effectively Serving the Changing Needs of DoD Missions

Cyber-Physical and Ultra-Large-Scale Systems

Research, Technology, and System Solutions

Ensuring Safety in Cyber-Physical Systems

In some key industries, such as defense, automobiles, medical devices, and the smart grid, the bulk of the innovations focus on cyber-physical systems. A key characteristic of cyber-physical systems is the close interaction of software components with physical processes, which impose stringent safety and time/space performance requirements on the systems. This article describes research and development we are conducting at the Software Engineering Institute to optimize the performance of cyber-physical systems without compromising their safety.

Cyber-physical systems are often safety-critical since violations of their requirements, such as missed deadlines or component failures, may have life-threatening consequences. For example, when a cyber-physical system in a car detects a crash, the airbag must inflate in less than 20 milliseconds to avoid severe injuries to the driver. Industry competitiveness, along with the urgency of fielding cyber-physical systems to meet rapidly evolving Department of Defense (DoD) mission needs, are increasingly pressuring manufacturers to implement cost and system performance optimizations without understanding their safety consequences. The impact of this lack of understanding on the commercial world can be seen in recent automotive recalls, delays in the delivery of new airplanes, and airplane accidents.

Although optimizing a cyber-physical system is hard, cost-reduction market pressures and small-form factors (e.g., small, remotely piloted aircraft [RPA]) often demand optimizations. An additional challenge faced by DoD cyber-physical systems is the scheduling of real-time tasks for which the amount of computation performed is not fixed but depends on the environment. For instance, the computation time of collision avoidance algorithms in RPA systems often varies in proportion to the objects the RPA finds in its path. This variation is hard to accommodate in traditional real-time scheduling theory, which assumes a fixed, worst-case execution time. Nonetheless, real-time scheduling is essential for RPAs and other autonomous systems that must function effectively in dynamic environments with limited human intervention.

As part of our research, we are investigating a safe “double-booking” of processing times between safety-critical and non-safety-critical tasks that can tolerate occasional timing failures (deadline misses). This double-booking approach helps reduce the over-allocation of processing resources needed to ensure the timing behavior of safety-critical tasks. Timing assurance is possible in conventional real-time systems by reserving sufficient processing time for tasks to execute for their worst-case execution time. The typical execution time of these tasks, however, is often less than the worst-case execution time, which occurs very rarely in practice. The difference between the worst-case and typical execution time of these tasks is thus considered an over-allocation.

Our approach takes advantage of over-allocation by packing safety-critical and non-safety-critical tasks together, letting the latter use the processing time that was over-allocated to the former. This approach essentially double-books processing time to both the safety- and non-safety-critical tasks. To assure the timing of the safety-critical tasks, however, whenever these tasks need to run for their worst-case execution time, we stop noncritical tasks. We identify this approach as an asymmetric protection scheme since it protects critical tasks from noncritical ones, but does not protect noncritical tasks from critical ones.

An example of where asymmetric protection can be applied is an automotive system. To continue with our earlier air bag example, a car’s air bag inflator has a task that continuously checks whether a crash has occurred. Of the 20 milliseconds allotted for airbag deployment, it may take only 5 milliseconds to conduct the check. If a crash has occurred, the airbag will continue to inflate during the remaining 15 milliseconds. If no crash has occurred, however, the remaining 15 milliseconds that the processor was reserved for this task will be available for non-safety-critical tasks, such as fuel efficiency, acceleration, and active suspension.

The deliverables from our project will include a modified version of the Linux operating system that implements the temporal protection scheme for mixed-criticality systems and the appropriate analysis algorithms to verify the timing behavior of the system. We will also develop optimization algorithms to maximize the utility that users can achieve from different applications available in the modified operating system. We are collaborating with Jeffrey Hansen of the Institute for Complex Engineered Systems, which is part of Carnegie Mellon University's (CMU) Carnegie Institute of Technology; John Lehoczky of CMU's Statistics Department; and Ragunathan (Raj) Rajkumar of the Electrical and Computer Engineering Department at CMU.

By Dionisio de Niz, Senior Member of the Technical Staff

Related Web Sites

www.contrib.andrew.cmu.edu/~dionisio/

For General Information

Research, Technology, and System Solutions

Toward Safe Optimization of Cyber-Physical Systems

Cyber-physical systems (CPS) are characterized by close interactions between software components and physical processes. These interactions can have life-threatening consequences when they include safety-critical functions that are not performed according to their time-sensitive requirements. For example, an airbag must fully inflate within 20 milliseconds (its deadline) of an accident to prevent the driver from hitting the steering wheel with potentially fatal consequences.

Unfortunately, the competition of safety-critical requirements with other demands to reduce the cost, power consumption, and device size also creates problems, such as automotive recalls, new aircraft delivery delays, and plane accidents. Our research leverages the fact that failing to meet deadlines doesn't always have the same level of criticality for all functions. For instance, if a music player fails to meet its deadlines, the sound quality may be compromised, but lives are not threatened. Systems whose functions have different criticalities are known as mixed-criticality systems. This article updates our earlier one to describe the latest results of our research on supporting mixed-criticality operations by giving more central processing unit (CPU) time to functions with higher value while ensuring critical timing guarantees.

During our research, we observed that different functions provide different amounts of utility or satisfaction to the user. For instance, a GPS navigation function may provide higher utility than a music player. Moreover, if we give more resources to these functions (for example, more CPU time) the utility obtained from them increases.

In general, however, the amount of utility obtained from additional resources does not grow forever, nor does it grow at a constant rate. The additional increment in utility for each additional unit of resource instead decreases to a point where the next increment in utility is insignificant. In such cases, it is often more important to dedicate additional computational resources to another function that is currently delivering lower utility and will deliver a larger increment in utility for the same amount of CPU time.

For example, assuming that we get a faster route to our destination if more CPU time is dedicated to the GPS functionality, it seems obvious that the first route we get from the GPS will give us the biggest increment in utility. If we lack enough CPU time (due to the execution of other critical functions) to run both the GPS and the music player, we will choose the GPS. We may even prefer to give more CPU time (if we discover that more time is available) to the GPS to help avoid traffic jams before we decide to run the music player. Letting the GPS run even longer to select a less traffic-clogged route, however, may give us less utility than running the music player.

At this point, we may prefer to start running the music player if we have more CPU time available. We thus change our allocation preference because the additional utility obtained by giving the GPS more CPU time is less than the utility obtained by giving the music player this time. This progressive decrease in the utility obtained as we give more resources to a function is known as diminishing returns, which can be used to allocate resources to ensure we obtain the maximum total utility possible considering all functions in the system.

Our research uses both the diminishing returns characteristics of low-criticality functions and criticality levels to implement a double-booking computation-time reservation scheme. Traditional real-time scheduling techniques consider the worst-case execution time (WCET) of the functions to ensure they always complete before their deadlines by reserving CPU time used only on the rare occasion that the WCET occurs. We take advantage of this fact and allocate the same CPU time for functions of lower criticality. When both functions request the CPU time reserved for both at the same time, we favor the higher-criticality function and let the lower-criticality function miss its deadline.

Our double-booking scheme is analogous to the strategies airlines use to assign the same seat to more than one person. In this case, the seat is given to the person with preferred status (e.g., "gold members"). Our project uses utility—in addition to criticality—to ensure that the CPU time that is double-booked is given to functions providing the largest utility in case of a conflict (both functions requesting the double-booked CPU time). Our double-booking scheme provides the following two benefits:

- It protects critical functions, ensuring that their deadlines are always met.
- It uses the unused time from the critical functions to run the noncritical functions that produce the highest utility.

Our research is aimed at providing real-time system developers with an analysis algorithm that accurately predicts system behavior when it is running (runtime). Developers use these algorithms during the design phase (design time) to test whether critical tasks will meet their deadlines (providing assurance) and how much overbooking is possible.

To evaluate the effectiveness of our scheme, we developed a utility degradation resilience (UDR) metric that quantifies the capacity of a CPS to preserve the utility derived from double-booking. This metric evaluates all possible conflicts that can happen due to double-booking and how much total utility is preserved after the conflict is resolved by deciding what function gets the double-booked CPU time and what functions are left without CPU time. The utility derived from the preserved functions is then summed to compute the total utility that a specific conflict resolution scheme can preserve.

In theory, a perfect conflict resolute scheme should preserve the maximum possible utility. In reality, however, decisions must be made ahead of time assuming that some critical functions will run for their WCET (even though they may not) to ensure that they finish before their deadlines. Unfortunately, if they execute for less time, it may already be too late to execute other functions.

Using the UDR metric, we compare our scheme against the Rate-Monotonic Scheduler (RMS) and a scheme called Criticality-As-Priority Assignment that uses the criticality as the priority. Our experiments showed we can recover up to 88 percent of the ideal utility that we could get if we could fully reclaim the unused time left by the critical functions and if we had perfect knowledge of exactly how much time each function needed to finish executing. In addition, we observed our double-booking scheme can achieve up to three times the UDR that RMS provides.

We implemented a design-time algorithm to evaluate the UDR of a system and generate the scheduling parameters for our runtime scheduler that performs the conflict resolutions of our overbooking scheme (deciding which function gets the overbooked CPU time). This scheduler was implemented in the Linux operating system as a proof of concept to evaluate the practicality of our mechanisms. To evaluate our scheme in a real-world setting, we used our scheduler in a surveillance unmanned aerial vehicle application using the Parrot A.R. Drone quadricopter with safety-critical functions (flight control) and two noncritical functions (video-streaming and vision-based object-detection functions).

Our results confirmed that we can recover more CPU cycles for noncritical tasks with our scheduler than with the fixed-priority scheduler (using rate-monotonic priorities) without causing problems to the critical tasks. For example, we avoided instability in the flight controller that can lead to the quadricopter turning upside down. In addition, the overbooking between the noncritical tasks performed by our algorithm allowed us to adapt automatically to peaks in the number of objects to detect (and hence execution time of the object detection function) by reducing the frames per second processed by the video-streaming function during these peaks.

In future work we are extending our investigation to multicore scheduling, for which we plan to apply our scheme to hardware resources (such as caches) shared across cores.

This research is done in collaboration with Jeffrey Hansen of Carnegie Mellon University (CMU), John Lehoczky of CMU's Statistics Department, and Ragunathan (Raj) Rajkumar and Anthony Rowe of the Electrical and Computer Engineering Department at CMU.

By Dionisio de Niz, Senior Member of the Technical Staff

Related Web Sites

www.contrib.andrew.cmu.edu/~dionisio/

For General Information

For information about the SEI and its products and services, contact
Customer Relations
Phone: 412-268-5800
FAX: 412-268-6257
info@sei.cmu.edu
www.sei.cmu.edu

Research, Technology, and System Solutions

Real-Time Scheduling on Heterogeneous Multicore Processors

Many Department of Defense computing systems—particularly cyber-physical systems—are subject to stringent size, weight, and power requirements. The quantity of sensor readings and functionalities is also increasing, and their associated processing must fulfill real-time requirements. This situation motivates the need for computers with greater processing capacity. For example, to fulfill the requirements of nano-sized unmanned aerial vehicles (UAVs), developers must choose a computer platform that offers significant processing capacity and use its processing resources to meet its needs for autonomous surveillance missions. This article discusses these issues and highlights our research that addresses them.

To choose a computer platform that offers greater capacity, it is necessary to observe the major trends among chip makers. Historically, advances in semiconductor miniaturization (a.k.a., Moore's Law) periodically yielded microprocessors with significantly greater clock speeds. Unfortunately, microprocessor serial processing speed is reaching a physical limit due to excessive power consumption. As a result, semiconductor manufacturers are now producing chips without increasing the clock speed, but instead are increasing the number of processor cores on a chip, which results in multicore processors. For nearly a decade, the use of homogeneous multicore processors (which are chips with identical processing cores) gave us some headroom in terms of power consumption and allowed us to enjoy greater computing capacity. This headroom is diminishing, unfortunately, and is about to vanish, forcing semiconductor manufacturers to seek new solutions.

We are currently witnessing a shift among semiconductor manufacturers from homogeneous multicore processors with identical processor cores to heterogeneous multicore processors. The impetus for this shift is that processor cores tailored to a specific class of applications behavior can offer much better power efficiency. AMD Fusion and NVIDIA Tegra 3 are examples of this shift. Intel Sandybridge, which has a graphics processor integrated onto the same chip as the normal processor, also reflects this shift.

In a heterogeneous multicore environment, the execution time of a software task depends on which processor core it executes on. For example, a software task performing computer graphics rendering, simulating physics, or estimating trajectories of flying objects runs much faster on a graphics processor than on a normal processor. Conversely, some software tasks are inherently sequential and cannot benefit from the graphics processor; they execute much faster on a normal processor. For example, a software task with many branches and no inherent parallelism runs much faster on a normal processor than on a graphics processor. Ideally, each task would be assigned to the processor where it executes with the greatest speed, but unfortunately the workload is often not perfectly balanced to the types of processor cores available.

Efficient use of processing capacity in the new generation of microprocessors therefore requires that tasks are assigned to processors intelligently. In this context, “intelligently” means that the resources requested by the program are the ones possessed by the processor. Moreover, the desire for short design cycles, rapid fielding, and upgrades necessitates that task assignment be done automatically—with algorithms and associated tools.

The Task Assignment Problem

The problem of assigning tasks to processors can be described as follows: A task (such as computer graphics rendering or a program determining whether the process half-or-triple-plus-one reaches one with a known starting value) is described with its processor utilization, but it has different processor utilizations for different processors. For example, if a given task is assigned to a graphics processor, then the task will have a utilization of 10 percent. If the task is assigned to a normal processor, the task will have a utilization of 70 percent. We are interested in assigning each task to exactly one processor such that for each processor, the sum of utilization of all tasks assigned to this processor will not exceed 100 percent. If we can find such an assignment, it is known that if tasks have deadlines described with the model implicit-deadline sporadic tasks—and if the scheduling algorithm Earliest-Deadline-First (EDF) is used—then all deadlines will be met at runtime (with a minor modification, we can also use rate-monotonic scheduling).

Previous Approaches for Task Assignment

The task assignment problem belongs to a class of problems that are computationally intractable, meaning that it is highly unlikely to design an algorithm that finds a good assignment and always runs fast. So we should either create an algorithm that always finds a good assignment or one that always runs fast. To design an algorithm that always finds a good assignment, we model task assignment as integer-linear programming (ILP) as follows:

Minimize z
subject to the constraints that for each
processor p: $x_{1,p} * u_{1,p} + x_{2,p} * u_{2,p} + \ldots + x_{n,p} * u_{n,p} <= z$
and
for each task i: $x_{i,1} + x_{i,2} + \ldots + x_{i,m} = 1$
and
for each pair (i,p) of task i and processor p: $x_{i,p}$ is either 0 or 1

In the optimization problem above, n is the number of tasks, m is the number of processors, and $u_{i,p}$ is the utilization of task i if it would be assigned to processor p. $x_{i,p}$ is a decision variable with the interpretation that it is 1 if task i is assigned to processor p and 0 otherwise.

Unfortunately, solving this integer linear program takes a long time.

To design an algorithm that always runs reasonably fast, there are several algorithms, as described in a research paper by Sanjoy K. Baruha, that transform the ILP into a linear program (LP) and then perform certain tricks. Although LPs runs faster than ILPs, they still have to solve an optimization problem, which can be time-consuming. To design algorithms that run faster, we would like to perform task assignment in a way that does not require solving LP.

Our Approach for Task Assignment

Previous work on task assignment for homogeneous multicore processors where all processor cores are identical is based on a framework called bin-packing heuristics. Such algorithms work approximately as follows:

```
1. Sort tasks according to some criterion.
2.   for each task do
3.     for each processor do
4.       if the task has not yet been assigned
         and it is possible to assign the task to
         the processor so that the sum of
         utilization of tasks on the processor
         does not exceed 100 percent then
5.         assign the task on the processor
6.       end if
7.     end for
8. end for
```

Our approach involves adapting bin-packing heuristics to heterogeneous multicore processors. We believe it is possible to modify the algorithm structure outlined above so we can also assign tasks to processors even when the utilization of a task depends on the processor to which it is assigned. One can show that bin-packing performs poorly if processors and tasks are not considered in any particular order. Specifically, for a set of tasks that could be assigned, such an approach can fail even when given processors that are "infinitely" faster. One of our main research challenges is therefore to determine how to sort tasks (Step 1) and in which order we should consider processors (in Step 3). We are evaluating our new algorithms in the following ways:

We plan to prove mathematically the performance of our new algorithms. Specifically, we are interested in proving that if it is possible to assign tasks to processors, then our algorithm will succeed in assigning tasks to a processor if a given processor is x times as fast. Given that x is our performance metric, the lower its value, the better.

We also plan to evaluate the performance of our algorithms by applying the algorithms on randomly generated task sets. This will demonstrate the typical behavior of the algorithms.

Conclusion

Most semiconductor manufacturers are shifting toward heterogeneous multicore processors to offer greater computing capacity while keep power consumption sufficiently low. But using a heterogeneous multicore efficiently for cyber-physical systems with stringent size, weight, and power requirements requires that tasks are assigned properly. This article has discussed the state of the art and summarized our ongoing work in this area.

By Bjorn Andersson
Senior Member of the Technical Staff

Related Web Sites

http://www.sei.cmu.edu/cyber-physical

For General Information

For information about the SEI and its products and services, contact
Customer Relations
Phone: 412-268-5800
FAX: 412-268-6257
info@sei.cmu.edu
www.sei.cmu.edu

Research, Technology, and System Solutions

Real-Time Scheduling on Heterogeneous Multicore Processors

Many Department of Defense computing systems—particularly cyber-physical systems—are subject to stringent size, weight, and power requirements. The quantity of sensor readings and functionalities is also increasing, and their associated processing must fulfill real-time requirements. This situation motivates the need for computers with greater processing capacity. For example, to fulfill the requirements of nano-sized unmanned aerial vehicles (UAVs), developers must choose a computer platform that offers significant processing capacity and use its processing resources to meet its needs for autonomous surveillance missions. This article discusses these issues and highlights our research that addresses them.

To choose a computer platform that offers greater capacity, it is necessary to observe the major trends among chip makers. Historically, advances in semiconductor miniaturization (a.k.a., Moore's Law) periodically yielded microprocessors with significantly greater clock speeds. Unfortunately, microprocessor serial processing speed is reaching a physical limit due to excessive power consumption. As a result, semiconductor manufacturers are now producing chips without increasing the clock speed, but instead are increasing the number of processor cores on a chip, which results in multicore processors. For nearly a decade, the use of homogeneous multicore processors (which are chips with identical processing cores) gave us some headroom in terms of power consumption and allowed us to enjoy greater computing capacity. This headroom is diminishing, unfortunately, and is about to vanish, forcing semiconductor manufacturers to seek new solutions.

We are currently witnessing a shift among semiconductor manufacturers from homogeneous multicore processors with identical processor cores to heterogeneous multicore processors. The impetus for this shift is that processor cores tailored to a specific class of applications behavior can offer much better power efficiency. AMD Fusion and NVIDIA Tegra 3 are examples of this shift. Intel Sandybridge, which has a graphics processor integrated onto the same chip as the normal processor, also reflects this shift.

In a heterogeneous multicore environment, the execution time of a software task depends on which processor core it executes on. For example, a software task performing computer graphics rendering, simulating physics, or estimating trajectories of flying objects runs much faster on a graphics processor than on a normal processor. Conversely, some software tasks are inherently sequential and cannot benefit from the graphics processor; they execute much faster on a normal processor. For example, a software task with many branches and no inherent parallelism runs much faster on a normal processor than on a graphics processor. Ideally, each task would be assigned to the processor where it executes with the greatest speed, but unfortunately the workload is often not perfectly balanced to the types of processor cores available.

Efficient use of processing capacity in the new generation of microprocessors therefore requires that tasks are assigned to processors intelligently. In this context, “intelligently” means that the resources requested by the program are the ones possessed by the processor. Moreover, the desire for short design cycles, rapid fielding, and upgrades necessitates that task assignment be done automatically—with algorithms and associated tools.

The Task Assignment Problem

The problem of assigning tasks to processors can be described as follows: A task (such as computer graphics rendering or a program determining whether the process half-or-triple-plus-one reaches one with a known starting value) is described with its processor utilization, but it has different processor utilizations for different processors. For example, if a given task is assigned to a graphics processor, then the task will have a utilization of 10 percent. If the task is assigned to a normal processor, the task will have a utilization of 70 percent. We are interested in assigning each task to exactly one processor such that for each processor, the sum of utilization of all tasks assigned to this processor will not exceed 100 percent. If we can find such an assignment, it is known that if tasks have deadlines described with the model implicit-deadline sporadic tasks—and if the scheduling algorithm Earliest-Deadline-First (EDF) is used—then all deadlines will be met at runtime (with a minor modification, we can also use rate-monotonic scheduling).

Previous Approaches for Task Assignment

The task assignment problem belongs to a class of problems that are computationally intractable, meaning that it is highly unlikely to design an algorithm that finds a good assignment and always runs fast. So we should either create an algorithm that always finds a good assignment or one that always runs fast. To design an algorithm that always finds a good assignment, we model task assignment as integer-linear programming (ILP) as follows:

Minimize z
subject to the constraints that for each processor p: $x_{1,p} * u_{1,p} + x_{2,p} * u_{2,p} + \ldots + x_{n,p} * u_{n,p} <= z$
and
for each task i: $x_{i,1} + x_{i,2} + \ldots + x_{i,m} = 1$
and
for each pair (i,p) of task i and processor p: $x_{i,p}$ is either 0 or 1

In the optimization problem above, n is the number of tasks, m is the number of processors, and $u_{i,p}$ is the utilization of task i if it would be assigned to processor p. $x_{i,p}$ is a decision variable with the interpretation that it is 1 if task i is assigned to processor p and 0 otherwise.

Unfortunately, solving this integer linear program takes a long time.

To design an algorithm that always runs reasonably fast, there are several algorithms, as described in a research paper by Sanjoy K. Baruha, that transform the ILP into a linear program (LP) and then perform certain tricks. Although LPs runs faster than ILPs, they still have to solve an optimization problem, which can be time-consuming. To design algorithms that run faster, we would like to perform task assignment in a way that does not require solving LP.

Our Approach for Task Assignment

Previous work on task assignment for homogeneous multicore processors where all processor cores are identical is based on a framework called bin-packing heuristics. Such algorithms work approximately as follows:

```
1. Sort tasks according to some criterion.
2.   **for** each task **do**
3.     **for** each processor **do**
4.       if the task has not yet been assigned
         and it is possible to assign the task to
         the processor so that the sum of
         utilization of tasks on the processor
         does not exceed 100 percent then
5.         assign the task on the processor
6.       end **if**
7.     end **for**
8. end **for**
```

Our approach involves adapting bin-packing heuristics to heterogeneous multicore processors. We believe it is possible to modify the algorithm structure outlined above so we can also assign tasks to processors even when the utilization of a task depends on the processor to which it is assigned. One can show that bin-packing performs poorly if processors and tasks are not considered in any particular order. Specifically, for a set of tasks that could be assigned, such an approach can fail even when given processors that are "infinitely" faster. One of our main research challenges is therefore to determine how to sort tasks (Step 1) and in which order we should consider processors (in Step 3). We are evaluating our new algorithms in the following ways:

We plan to prove mathematically the performance of our new algorithms. Specifically, we are interested in proving that if it is possible to assign tasks to processors, then our algorithm will succeed in assigning tasks to a processor if a given processor is x times as fast. Given that x is our performance metric, the lower its value, the better.

We also plan to evaluate the performance of our algorithms by applying the algorithms on randomly generated task sets. This will demonstrate the typical behavior of the algorithms.

Conclusion

Most semiconductor manufacturers are shifting toward heterogeneous multicore processors to offer greater computing capacity while keep power consumption sufficiently low. But using a heterogeneous multicore efficiently for cyber-physical systems with stringent size, weight, and power requirements requires that tasks are assigned properly. This article has discussed the state of the art and summarized our ongoing work in this area.

By Bjorn Andersson
Senior Member of the Technical Staff

Related Web Sites

http://www.sei.cmu.edu/cyber-physical

For General Information

For information about the SEI and its products and services, contact
Customer Relations
Phone: 412-268-5800
FAX: 412-268-6257
info@sei.cmu.edu
www.sei.cmu.edu

Research, Technology, and System Solutions

Regression Verification for Real-Time Embedded Software Systems

The Department of Defense relies heavily on mission- and safety-critical real-time embedded software systems (RTESs), which play a crucial role in controlling systems ranging from airplanes and cars to infusion pumps and microwaves. Since RTESs are often safety critical, they must undergo an extensive (and often expensive) certification process before deployment. This costly certification process must be repeated after any significant change to the RTES, such as migrating a single-core RTES to a multicore platform, significant code refactoring, or performance optimizations. Our initial approach to reducing recertification effort focused on the parts of a system whose behavior was affected by changes using a technique called regression verification, which involves deciding the behavioral equivalence of two closely related programs. This article describes our latest research in this area, specifically our approach to building regression verification tools and techniques for static analysis of RTESs.

Although there are many types of RTESs, we concentrate on a class of periodic programs, which are concurrent programs that consist of tasks that execute periodically. The tasks are assigned priorities based on their frequency (higher frequency = higher priority). The RTES executes the tasks using a priority-based preemptive scheduler. Each execution of a task is called a job. Thus, from the perspective of the scheduler, a system's execution is a constant periodic stream of jobs of different priorities. In this article, we use RTES to mean periodic programs.

In the beginning of the project, we assumed that automated verification techniques (such as static analysis and model checking) for single-core RTESs could be adapted for regression verification since these techniques have been used for sequential single-core programs. After conducting an initial survey, however, we found that existing automated verification techniques that apply directly to a program source (rather than to a manual abstract model) are not applicable to periodic programs. We therefore changed our original approach to extend static analysis to regression verification in the setting of multicore RTES in two ways. First, in Phase 1 of our project we developed a new static analysis technique for reasoning about bounded executions of periodic programs. Second, in Phase 2 we extended regression verification to multithreaded programs, of which periodic programs are a restricted subset.

Phase 1: Time-Bounded Verification of Periodic Programs

In the first part of our work, we developed an approach for time-bounded verification of safety properties (user-specified assertions) of periodic programs written in the C programming language. Time-bounded verification is the problem of deciding whether a given program does not violate any user-specified assertions in a given time interval. Time-bounded verification makes sense for RTESs because of their intimate dependence on real-time behavior. The inputs to our approach are (1) a periodic program C, (2) a safety property expressed via an assertion A embedded in C, (3) an initial condition Init of C, and (4) a time-bound W. The output is either a counter-example trace showing how C violates an assertion A, or a message saying that the program is safe because there is no execution that triggers any user-specified assertions.

Our solution to time-bounded verification is based on sequentialization, which involves reducing verification of a current program P to verification of a (non-deterministic) sequential program P′. A key feature of our approach is that P′ is linear in the size of P, which means the translation step is not computationally intensive and adds little overhead to the verification effort. The scalability of our approach is therefore mostly driven by the scalability of the underlying analysis engine, and our approach automatically benefits from constant improvements in the verification area.

Our work builds on previous sequentialization work for context-bounded analysis (CBA) and bounded model checking (BMC). Our approach differs from prior work, however, since it bounds the actual execution time of the program, which is more natural to the designer of an RTES than a bound on the number of context switches (as done in CBA) or a bound on the number of instructions executed (as in BMC). We bound the execution time by translating the input time-bound W in our model to a bound on the number of jobs. This translation is a natural consequence of the fact that the tasks are periodic and are therefore activated a finite number of times within W.

We implemented our approach in a tool called REK. REK supports C programs with tasks, priorities, priority ceiling locks, and shared variables. It takes a concurrent periodic program that cannot be analyzed with standard tools for sequential verification and converts it to become analyzable with such tools. Although in principle REK is compatible with any analyzer for bounded (loop- and recursion-free) C programs, in practice we rely on the CBMC tool by Daniel Kroening, which is one of the first and most mature bounded model checkers for C. CBMC can automatically analyze substantial C programs by encoding assertion violation to Boolean satisfiability queries. CBMC is a mature and robust tool that has been extensively applied to many industrial problems.

How REK Works

The analysis problem that REK is designed to solve is to check that a given periodic program is safe under all legal scheduling of tasks. REK solves a time-bounded version of this problem, for example, whether the program is safe in the first 100 ms, 200 ms, 300 ms, and so forth, starting from some user-specified initial condition. A time-bounded verification makes sense in the context of periodic programs since their execution can be naturally partitioned by time intervals. Of course, in practice, unbounded verification would be preferred, so we are working on extending REK in this direction.

We briefly summarize the sequentialization step done by REK. First, we divide a time-bounded execution into execution rounds (or rounds, for short). The execution starts in Round 0; a new round starts (and the old one stops) whenever a job of some task finishes. An execution with X jobs therefore requires X execution rounds. The sequentialization step simulates execution of each round independently and then combines them (using nondeterministic choice) into a single legal execution.

In addition to the basic sequentialization, we extended REK with the following features to achieve scalability to realistic programs:

Partial order reduction is a set of techniques used in model checking to reduce the number of interleavings that must to be explored in a concurrent system. For example, if there are two independent actions *a* and *b*, then only one of the two executions “*a* followed by *b*” or “*b* followed by *a*” must be explored since they both lead to the same destination state. Although there are many approaches for partial order reduction in explicit state model checking (as opposed to symbolic model checking used in this work), extending them to symbolic verification is an area of active research. In REK, we developed a new partial order reduction technique that restricts explored executions only to those in which a read statement is preempted by a write statement to the same variable, or a write is preempted by a read or a write. This reduction eliminates many unnecessary interleavings and cuts the search space significantly. Our experiments show that the reduction is quite effective in practice.

A limitation to our approach is that it does not keep track of the actual execution time of each instruction, each job, and each task. As such, it is an over-approximation since it explores more executions than actually possible and can produce a false positive by producing a counter-example trace that is not possible on a given hardware architecture due to timing restrictions. To reduce the number of false positives, we further constrain our sequentialization by the information that can be inferred from schedulability analysis. Thus, if a periodic program is schedulable, it satisfies the rate monotonic analysis equations. Those equations can be used to compute an upper bound on the number of times any given low-priority job can be preempted by any given high-priority job. We call this the *preemption bound*, which REK uses to further reduce the number of interleavings by keeping track how many times one task preempts another and ensuring that this value never exceeds the preemption bound for the jobs of that task.

To deal with practical periodic programs, REK provides support for two types of commonly used *lock primitives*. In particular, it supports preemption locks (preemptions are disabled when the lock is held) and priority ceiling locks (preemption by any task with lower priority than the lock is disabled when the lock is held). We are extending REK to support the third common type of locks, priority-inheritance locks (regular blocking locks, but the priority of a low-priority task that holds a lock *l* is increased if a high-priority task is waiting for *l*).

As part of our research, we created a model problem using the NXTway-GS, which is a two-wheeled, self-balancing robot that responds to Bluetooth commands. The robot uses a gyroscope to balance itself upright by applying power to left and right wheels. It also uses a sonar sensor so that when it comes to an obstacle, like a wall or ditch, it can back up. We have used REK to verify and fix several communication consistency properties between the tasks of the robot.

Phase 2: Regression Verification for Multi-threaded Programs

In the second phase of our work, we examined regression verification for multi-threaded programs. We believe that that once we have regression verification for multithreaded programs, we can adapt it to periodic programs as well.

Every instance of regression verification is based on some underlying notion of equivalence. The equivalence notion for single-threaded software is called partial equivalence: two functions are partially equivalent if they produce the same output for the same input. A multithreaded program, conversely, is not partially equivalent to itself by the above definition since the same input can lead to different outputs due to scheduling choices. Our first challenge therefore involved creating a notion of equivalence for multithreaded software.

Our second challenge was to come up with the right notion of decomposition to establish equivalence of programs from equivalence of their functions. Equivalence of sequential programs is done using Input/Output equivalence. Two sequential programs are equivalent if it is possible to show that their corresponding functions have the same Input/Output behavior (produce the same output given the same input). In the case of multithreaded programs, however, functions from different threads of a single program affect one another, making simple decomposition at the level of functions much harder because it must take interference from other threads into account.

To check whether two multithreaded programs are partially equivalent (P = P′) we use a proof rule consisting of a set of premises and a conclusion. Each premise establishes the partial equivalence of a pair of functions f and f′ from P and P′, respectively. A premise is established by verifying a single-threaded program.

As part of this work, we developed two separate proof rules:

The first rule attempts to show equivalence of two programs by showing that their corresponding functions are Input/Output equivalent (produce the same output for a given input) under arbitrary interference, where “interference” means that the value of shared variables can change between execution of instructions of a thread. This rule is “strong” (not widely applicable on many equivalent programs) because in practice the functions must be equivalent only in the context of the given program and not under arbitrary interference.

The second rule improves on the first rule by attempting to show that two programs are equivalent by restricting interference to what is consistent with the other functions in the program. For example, if there is no other function in a program that can affect a global variable *x*, then no interference that modifies *x* is considered. This rule is “weaker” (more widely applicable) than the first one but is computationally harder to automate.

Conclusion

The abilities to statically reason about correctness of periodic programs and to perform regression verification add the following key capabilities to an RTES developer’s toolbox:

- ability to check prior to deployment that the program does not violate its assertions
- ability to check that top-level application programming interfaces (APIs) are not affected by low-level refactoring or performance optimizations
- ability to check that new APIs are backward compatible with old APIs
- ability to perform impact analysis to determine which function may possibly be affected by a given source code change and which unit tests must be repeated

We believe these capabilities can lower the cost of developing RTESs while increasing their reliability and trustworthiness.

By Arie Gurfinkel
Senior Member of the Technical Staff

Related Web Sites

www.sei.cmu.edu/cyber-physical
www.andrew.cmu.edu/~arieg/Rek

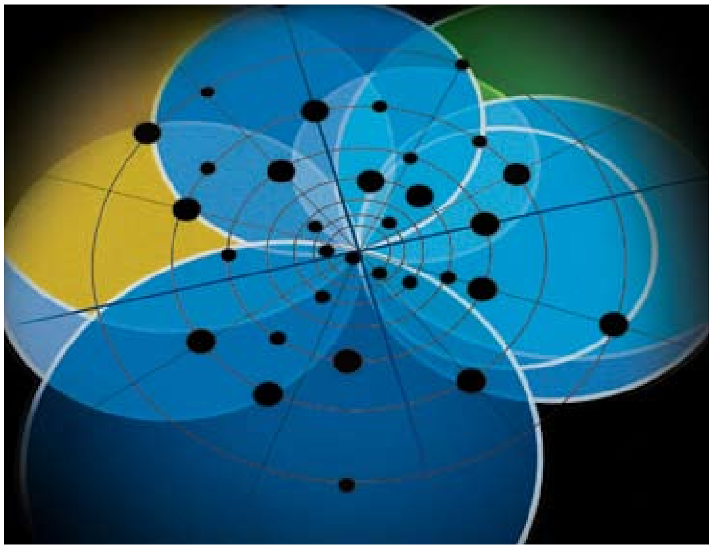
For General Information

For information about the SEI and its products and services, contact
Customer Relations
Phone: 412-268-5800
FAX: 412-268-6257
customer-relations@sei.cmu.edu
www.sei.cmu.edu

Software Engineering Institute | Carnegie Mellon

Ultra-Large-Scale Systems

The Software Challenge of the Future



Ultra-Large-Scale Systems: The Software Challenge of the Future is the product of a 12-month study of ultra-large-scale (ULS) systems software. The study brought together experts in software and other fields to answer a question posed by the Office of the Assistant Secretary of the U.S. Army (Acquisition, Logistics & Technology): "Given the issues with today's software engineering, how can we build the systems of the future that are likely to have billions of lines of code?" Increased code size brings with it increased scale in many dimensions, posing challenges that strain current software foundations. The report details a broad, multi-disciplinary research agenda for developing the ultra-large-scale systems of the future.

What are ULS systems?

The U. S. Department of Defense (DoD) has a goal of information dominance—to achieve and exploit superior collection, fusion, analysis, and use of information to meet mission objectives. This goal depends on increasingly complex systems characterized by thousands of platforms, sensors, decision nodes, weapons, and warfighters connected through heterogeneous wired and wireless networks. These systems will push far beyond the size of today's systems and systems of systems by every measure: number of lines of code; number of people employing the system for different purposes; amount of data stored, accessed, manipulated, and refined; number of connections and interdependencies among software components; and number of hardware elements. They will be ultra-large-scale (ULS) systems.

How are ULS systems different?

The sheer scale of ULS systems will change everything. ULS systems will necessarily be decentralized in a variety of ways, developed and used by a wide variety of stakeholders with conflicting needs, evolving continuously, and constructed from heterogeneous parts. People will not just be users of a ULS system; they will be elements of the system. Software and hardware failures will be the norm rather than the exception. The acquisition of a ULS system will be simultaneous with its operation and will require new methods for control. These characteristics are beginning to emerge in today's DoD systems of systems; in ULS systems they will dominate. Consequently, ULS systems will place unprecedented demands on software acquisition, production, deployment, management, documentation, usage, and evolution practices.

Challenges of ULS systems

Fundamental gaps in our current understanding of software and software development at the scale of ULS systems present profound impediments to the technically and economically effective achievement of the DoD goal of deterrence and dominance based on information superiority. These gaps are strategic, not tactical. They are unlikely to be addressed adequately by incremental research within established categories. Rather, we require a broad new conception of both the nature of such systems and new ideas for how to develop them. We will need to look at them differently, not just as systems or systems of systems, but as socio-technical ecosystems. We will face fundamental challenges in the design and evolution, orchestration and control, and monitoring and assessment of ULS systems. These challenges require breakthrough research.

The SEI's ULS research agenda

We propose a ULS systems research agenda for an interdisciplinary portfolio of research in at least the following areas:

- Human Interaction: involves anthropologists, sociologists, and social scientists conducting detailed socio-technical analyses of user interactions in the field, with the goal of understanding how to construct and evolve such socio-technical systems effectively.

- Computational Emergence: explores the use of methods and tools based on economics and game theory (e.g., mechanism design) to ensure globally optimal ULS system behavior and explores metaheuristics and digital evolution to augment the cognitive limits of human designers.

- Design: broadens the traditional technology-centric definition of design to include people and organizations; social, cognitive, and economic considerations; and design structures such as design rules and government policies.
- Computational Engineering: focuses on evolving the expressiveness of representations to accommodate the semantic diversity of many languages and focuses on providing automated support for computing the evolving behavior of components and their compositions.
- Adaptive System Infrastructure: investigates integrated development environments and runtime platforms that will support the decentralized nature of ULS systems as well as technologies, methods, and theories that will enable ULS systems to be developed in their deployment environments.
- Adaptable and Predictable System Quality: focuses on how to maintain quality in a ULS system in the face of continuous change, ongoing failures, and attacks and focuses on how to identify, predict, and control new indicators of system health (akin to the U.S. gross domestic product) that are needed because of the scale of ULS systems.
- Policy, Acquisition, and Management: focuses on transforming acquisition policies and processes to accommodate the rapid and continuous evolution of ULS systems by treating suppliers and supply chains as intrinsic and essential components of a ULS system.

The proposed research does not supplant current, important software research but rather significantly expands its horizons. Moreover, because we are focused on systems of the future, we have purposely avoided couching our descriptions in terms of today's technology. The envisioned outcome of the proposed research is a spectrum of technologies and methods for developing these systems of the future, with national-security, economic, and societal benefits that extend far beyond ULS systems themselves.

Though our research agenda does not prescribe a single, definitive roadmap, we offer three structures that suggest ways to cluster and prioritize groups of research areas mapping the research areas and topics to (1) specific DoD missions and required capabilities, (2) DoD research funding types required to support them, and (3) estimates of the relative starting points of the research. These structures can then be used to define one or more roadmaps that could lead to one or more ULS systems research programs or projects.

Recommendations

As a first step, we recommend the funding and establishment of a ULS System Research Startup Initiative, which over the course of the next two years would, among other things

- work with others to conduct new basic research in key areas
- foster the growth of a community of informed stakeholders and researchers
- formulate and issue an initial Broad Agency Announcement (BAA) to attract researchers with proven expertise in the diverse set of disciplines (e.g., software engineering, economics, human factors, cognitive psychology, sociology, systems engineering, and business policy) that are collectively required to meet the challenge of ULS systems

The United States needs a program that will fund the software research required to sustain ongoing transformations in national defense and achieve the DoD goal of information dominance. The key challenge is the decision to move forward. The ULS System Research Agenda presented in Ultra-Large-Scale Systems: The Software Challenge of the Future provides the starting point for the path ahead.

If you would like more information about ULS systems and the ULS Systems Study, please contact Linda Northrop at lmn@sei.cmu.edu.

www.sei.cmu.edu/uls/

Edge-Enabled Tactical Systems

Research, Technology, and System Solutions

Equipping the Soldier with End-User Programming

Whether soldiers are on the battlefield or providing humanitarian relief effort, they need to capture and process a wide range of text, image, and map-based information. To support soldiers in this effort, the Department of Defense (DoD) is beginning to equip soldiers with smartphones to allow them to manage that vast array and amount of information they encounter while in the field. Whether the information gets correctly conveyed up the chain of command depends, in part, on the soldier's ability to capture accurate data in the field. This article, a follow-up to our initial one, describes our work on creating a software application for smartphones that allows soldier end-users to program their smartphones to provide an interface tailored to the information they need for a specific mission.

The software we developed is constructed primarily in Java and operates on an Android platform. We used an object database (DB 4.0) as the underlying data store because it provides flexible and powerful application programming interfaces that simplified our implementation. For performance reasons, our application is a native Android app—it's not running on a browser of an Android smart phone.

Our app—called eMONTAGE (Edge Mission-Oriented Tactical App Generator)—allows a soldier to build customized interfaces that support the two basic paradigms that are common to smartphones: maps and lists. For example, soldiers could build interfaces that allow them to construct a list of friendly community members including names, affiliations with specific groups, information about whether the person speaks English, and the names of the person's children. If soldiers also specify a GPS location in the customized interface they construct, the location of the friendly community members could be plotted on a map. Likewise, a soldier could build other customized interfaces that capture specific aspects of a threatening incident, or the names and capabilities of nongovernmental organizations (NGOs) responding to a humanitarian crisis.

Challenges We Encountered

The software we built is intended for soldiers who are well versed in their craft but are not programmers. While we are still conducting user testing, after we developed a prototype, we asked several soldiers to provide feedback. Not surprisingly, we found that soldiers who are Android users and relatively young (i.e., digital natives) quickly learned the software programming application and could use it to build a new application on-site. Conversely, non-digital natives had a harder time. Since our goal is to make our software accessible to every soldier, we are simplifying, revising, and improving the user interface.

As with any device used by our military, security is a key concern. Through our work with the Defense Advanced Research Projects Agency's Transformative Apps program in the Information Innovation office, we can take advantage of the security strategies they conceive and implement. We are also working to address challenges associated with limited bandwidth and battery consumption in this work and other work at the Software Engineering Institute.

Another area of our work involves enabling our software to connect to back-end data sources that the DoD uses. For example, a soldier on patrol may need to connect to TiGR and other information systems to access current information about people, places, and activities in an area. Our software will enable these soldiers to build customized interfaces to such data sources by selecting fields for display on the phone and by extending the information provided by these sources with additional, mission-specific information. This capability will provide mash-ups that support soldiers by capturing multiple sources of information for display and manipulation. Once our full capability is available in spring 2012, it will become much easier to build phone interfaces to new data sources and extend these interfaces with additional information.

Looking to the Future

Currently, eMONTAGE can handle the basic information types that are available on an Android phone, including images, audio, and data. Technologies like fingerprint readers and chemical sensors are being miniaturized and will likely be incorporated into future handheld devices. With each new technology, we'll need to add that basic type to our capability. Fortunately, this is a relatively straightforward programming operation, but it does require engineering expertise. As a new type becomes available, professional engineers will add it to eMONTAGE, thereby making the type available to soldiers who may have little or no programming expertise.

Our current focus is on ensuring that the software is reliable and does not fail, but we are also looking to extend it to provide features that we believe are essential, such as better support for collections of objects. For example, soldiers may need to classify a single individual into different groups: a family member, translator, or member of an organization. Each group is a collection. Soldiers will have the ability to list and search through collections (e.g., list all members of an NGO who work for Doctors Without Borders) and plot the members of a collection on a map (e.g., display all members of Doctors Without Borders who are within 10 miles of my current position).

While we can provide access to military iconology, eMONTAGE is not DoD-specific by design. This application can be used by other government organizations—or even NGOs— that want a user-customizable way to capture information about any variety of people, places, and things and share this information effectively in the enterprise.

Part of our ongoing research involves testing our applications with soldiers through the Naval Postgraduate School's Center for Network Innovation and Experimentation (CENETIX). In our initial tests with the soldiers, they told us what capabilities they need and what did not work. These collaborations tie our work firmly into both the research and military communities and keep us focused on providing a useful and cutting-edge capability. In addition to continuing our collaboration with CENETIX, we are working with Dr. Brad Myers of the Carnegie Mellon University Human Computer Interaction Institute. Dr. Myers is helping us define an appropriate interface for soldiers to use the handheld software in the challenging situations they face.

By Edwin Morris, Senior Member of the Technical Staff

Related Web Sites

http://blog.sei.cmu.edu/post.cfm/a-new-approach-for-handheld-devices-in-the-military

For General Information

For information about the SEI and its products and services, contact Customer Relations
Phone: 412-268-5800
FAX: 412-268-6257
info@sei.cmu.edu
www.sei.cmu.edu

Research, Technology, and System Solutions

A New Approach for Handheld Devices in the Military

Many people today carry handheld computing devices to support their business, entertainment, and social needs in commercial networks. The Department of Defense (DoD) is increasingly interested in having soldiers carry handheld computing devices to support their mission needs in tactical networks. Not surprisingly, however, conventional handheld computing devices (such as iPhone or Android smartphones) for commercial networks differ in significant ways from handheld devices for tactical networks. For example, conventional devices and the software that runs on them do not provide the capabilities and security needed by military devices, nor are they configured to work over DoD tactical networks with severe bandwidth limitations and stringent transmission security requirements. This article describes exploratory research we are conducting at the Software Engineering Institute (SEI) to (1) create software that allows soldiers to access information on a handheld device and (2) program the software to tailor the information for a given mission or situation.

To motivate the need for tactical handheld devices, imagine a U.S. soldier on patrol, deployed abroad, and walking into an unfamiliar village. Many pieces of information would be useful to that soldier in that situation. For example, it would be useful to know who the village elders are and to have pictures to identify them. It would also be useful to access information about previous improvised explosive device (IED) attacks, reports detailing the results of other contact that soldiers have had with villagers, and whether any friendly villagers speak English. We face the following challenges when creating software for tactical handheld computing devices that can provide this information:

- *Developing applications that can support the full range of military missions.* In recent years, soldiers have provided humanitarian assistance to victims of natural disasters in Haiti and countries in Asia, patrolled our country's borders, protected global waterways from piracy, and performed many types of military operations in Iraq and Afghanistan. These missions are sufficiently diverse that a one-size-fits-all software solution is not practical. For example, consider the different goals of clearing a route in a combat zone versus delivering humanitarian supplies in a relief effort or the different information required to protect from IED attacks versus treat a critically ill child. Not only is different information required, but also the rules for sharing it can vary. In a combat environment, security concerns require limiting access, while information in a relief mission may be shareable with nongovernmental organizations responding to the crisis.
- *Processing large amounts of data available through the rapid computerization and internetworking of various military missions.* For example, the military employs hundreds of unmanned aerial vehicles that generate large amounts of data. There are also increases in the number of sensors, such as auditory, biological, chemical, and nuclear, that are network enabled. All the data generated from these devices makes it hard to pinpoint the right information for a given mission and situation.

Our goal is to ensure the capabilities provided on tactical handheld computing devices are flexible enough to allow solders to control the amount and type of data that they receive and adaptive enough to meet the needs of particular missions. To achieve this goal, we are exploring the integration of end-user programming techniques, active data filtering and formatting, and confidence-building strategies. End-user programming techniques enable soldiers to program software on tactical handheld devices without requiring them to be professional software developers. Filtering incoming information and displaying it in intuitive formats helps avoid inundating soldiers on patrol with too much data. Confidence-building strategies promote trust that applications programmed by soldiers work correctly and safely. We are currently developing software for Android devices, but the fundamental concepts are applicable to other mobile platforms as well.

A key concern is designing software that has an intuitive and simple-to-use interface since the soldiers customizing these capabilities are not programmers; they are war fighters. The software we build must therefore help them readily find and assemble the types of information they need. It should reduce the soldier's workload by filling in (auto-complete) as much information for the soldier as possible. The software should require soldiers to learn only a few different types of screens (for example, screens for entering data and for establishing filters should be substantially the same). In addition, confidence-building feedback should be integrated into the interface so that soldiers are sure that what they build will work and are informed early if it will not.

Our work also focuses on ensuring that the information—whether from central command or a local unit—makes its way quickly and efficiently to the handheld computing device used by soldiers. For example, user-programmable data filtering allows soldiers to specify what information is important. Likewise, optimized protocol implementations ensure this information is exchanged quickly.

Last year, we conducted a research project that involved taking a service-oriented architecture approach to provide real-time situational awareness data to Android smartphones. We worked with soldiers through the Naval Postgraduate School's Center for Network Innovation and Experimentation (CENETIX) to test our applications. They told us what capabilities they need, and what did not work. These collaborations tie our work firmly into both the research and military communities and keep us focused on providing a useful and cutting-edge capability. In addition to continuing our collaboration with CENETIX, we are working with Dr. Brad Myers of the Carnegie Mellon University Human Computer Interaction Institute. Dr. Myers is helping us define an appropriate interface for soldiers to use the handheld software in the challenging situations they face.

By Edwin Morris, Senior Member of the Technical Staff

Related Web Sites

http://blog.sei.cmu.edu/post.cfm/a-new-approach-for-handheld-devices-in-the-military

For General Information

For information about the SEI and its products and services, contact
Customer Relations
Phone: 412-268-5800
FAX: 412-268-6257
info@sei.cmu.edu
www.sei.cmu.edu

Research, Technology, and System Solutions

Cloud Computing for the Battlefield

The Department of Defense (DoD) is increasingly interested in having soldiers carry handheld mobile computing devices to support their mission needs. Soldiers can use handheld devices to help with various tasks, such as speech and image recognition, natural language processing, decision making, and mission planning. Three challenges, however, present obstacles to achieving these capabilities. The first challenge is that mobile devices offer less computational power than a conventional desktop or server computer. A second challenge is that computation-intensive tasks, such as image recognition or even global positioning systems, take heavy tolls on battery power. The third challenge is dealing with unreliable networks and bandwidth. This article explores our research to overcome these challenges by using cloudlets, which are localized, lightweight servers running one or more virtual machines (VMs) on which soldiers can offload expensive computations from their handheld mobile devices, thereby providing greater processing capacity and conserving battery power.

Leveraging external resources to augment the capabilities of resource-limited mobile devices is a technique commonly known as cyber-foraging. The use of VM technology provides greater flexibility in the type and platform of applications and also reduces setup and administration time, which is critical for systems at the tactical edge. The term *tactical edge* refers to systems used by soldiers or first responders that are close to a mission or emergency executing in environments characterized by limited resources in terms of computation, power, and network bandwidth, as well as changes in the status of the mission or emergency.

Cloudlets are located within proximity of handheld devices that use them, thereby decreasing latency by using a single-hop network and potentially lowering battery consumption by using WiFi instead of broadband wireless, which consumes more energy. For example, a cloudlet might run in a Tactical Operations Center (TOC) or a Humvee. From a security perspective, cloudlets can use WiFi networks to take advantage of existing security policies, including access from only specific handheld devices and encryption techniques.

Related work on offloading computation to conserve battery power in mobile devices relies on the conventional Internet or environments that tightly couple applications running on handheld devices and servers on which computations are offloaded. In contrast, cloudlets decouple mobile applications from the servers. Each mobile app has a client portion and an application overlay corresponding to the computation-intensive code invoked by the client. On execution, the overlay is sent to the cloudlet and applied to one of the VMs running in the cloudlet, which is called dynamic VM synthesis. The application overlay is pre-generated by calculating the difference between a base VM and the base VM with the computation-intensive code installed. The only coupling that exists between the mobile app and the cloudlet is that the same version of the VM software on which the overlay was created must be used. Since no application-specific software is installed on the server, there is no need to synchronize release cycles between the client and server portions of apps, which simplifies the deployment and configuration management of apps in the field.

Dynamic VM synthesis is particularly useful in tactical environments characterized by unreliable networks and bandwidth, unplanned loss of cyber foraging platforms, and a need for rapid deployment. For example, imagine a scenario where a soldier needs to execute a computation-intensive app configured to work with cloudlets. At runtime, the app discovers a nearby cloudlet located on a Humvee and offloads the computation-intensive portion of code to it. Due to enemy attacks, network connectivity, or exhaustion of energy sources on the cloudlet, however, the mobile app is disconnected from the cloudlet. The mobile app can then locate a different cloudlet (e.g., in a TOC) and—due to dynamic VM synthesis—can have the app running in a short amount of time, with no need for any configuration on the app or the cloudlet. This flexibility enables the use of whatever resources become opportunistically available, as well as replacement of lost cyber-foraging resources and dynamic customization of newly acquired cyber-foraging resources.

As part of our research, we are focusing on face recognition applications. Thus far we have created an Android-based facial recognition app that performs the following actions:

1. It locates a cloudlet via a discovery protocol.
2. It sends the application overlay to the cloudlet where dynamic VM synthesis is performed.
3. It captures images and sends them to the facial recognition server code that now resides in the cloudlet.
4. The application overlay is a facial recognition server written in C++ that processes images from a client for training or recognition purposes. When in recognition mode, it returns coordinates for the faces it recognizes as well as a measure of confidence. The first version of the cloudlet is a simple HTTP server that receives the application overlay from the client, decrypts the overlay, decompresses the overlay, and performs VM synthesis to dynamically set up the cloudlet.

The first phase of our work has focused on creating the cloudlet prototype described above. In the second phase, we will conduct measurements to see if computations in a cloudlet provide significant reductions in device battery power. In addition, we will gather measurements related to bandwidth consumption of overlay transfer and VM synthesis to focus on optimization of cloudlet setup time. Assuming we are successful, our third phase will create a cloudlet in the RTSS Concept Lab to explore other ways to take computation to the tactical edge.

As part of our research, we are collaborating with Mahadev Satyanarayanan, the creator of the cloudlet concept and a faculty member at Carnegie Mellon University's School of Computer Science.

By Grace Lewis, Senior Member of the Technical Staff

Related Web Sites

www.sei.cmu.edu/sos/research/cloudcomputing
www.sei.cmu.edu/library/abstracts/webinars/Cloud-Computing.cfm
http://blog.sei.cmu.edu/archives.cfm/category/cloud-computing

For General Information

For information about the SEI and its products and services, contact
Customer Relations
Phone: 412-268-5800
FAX: 412-268-6257
info@sei.cmu.edu
www.sei.cmu.edu

Research, Technology, and System Solutions

Cloud Computing at the Tactical Edge

Cloudlets, which are lightweight servers running one or more virtual machines (VMs), allow soldiers in the field to offload resource-consumptive and battery-draining computations from their handheld devices to nearby cloudlets. This architecture decreases latency by using a single-hop network and potentially lowers battery consumption by using WiFi instead of broadband wireless. This article extends our original one by describing how we are using cloudlets to help soldiers perform various mission capabilities more effectively, including facial, speech, and imaging recognition, as well as decision making and mission planning.

An initial goal of our research was to create a prototype application that located cloudlets within close proximity of handheld devices using them. We initially focused on off-loading computations to cloudlets to extend device battery life. In addition to this benefit, we also found cloudlets significantly reduce the amount of time needed to deploy applications to handheld devices because clients are not tied to a specific server that can take a long time to provision in tactical environments.

Our work together with Mahadev “Satya” Satyanarayanan (the creator of the cloudlet concept and a faculty member at Carnegie Mellon's School of Computer Science) originally focused on face recognition applications as an example of a computation-intensive mission capability. Thus far we have created an Android-based facial recognition application that

- locates a cloudlet via a discovery protocol
- sends the application overlay to the cloudlet, where dynamic VM synthesis is performed
- captures the images and sends them to the facial recognition server code that now resides in the cloudlet

In the context of cloudlets, the application overlay corresponds to the computation-intensive code invoked by the client, which in this case is the face recognition server written in C++, and processes images from a handheld device client for training or recognition purposes. On execution, the overlay is sent to the cloudlet and applied to one of the VMs running in the cloudlet, which is called *dynamic VM synthesis*. The application overlay is pre-generated by calculating the difference between a base VM and the base VM with the computation-intensive code installed.

The first version of the cloudlet we created is a simple HTTP server. When this server receives the application overlay from the client, it decrypts and decompresses the overlay and performs VM synthesis to configure the cloudlet dynamically. It subsequently returns coordinates for the faces it recognizes, along with a measure of confidence to the client device.

Constructing the Cloudlet Prototype

The original cloudlet prototype built by Satya’s team used a simple Virtual Network Computer (VNC) client to see what was executing inside the VM. Our cloudlet prototype extended Satya’s work to use a thick mobile client that provides a better user experience for users at the edge and allows incorporation of sensor information that would not be possible with the original VNC cloudlet approach. We constructed this prototype in the Software Engineering Institute’s Concept Lab.

Our design was tricky because the face recognition client needs to know the IP address and the port on which the face recognition server is listening so that it can connect to it. The client uses an HTTP request to start the cloudlet setup and expects an HTTP response from the cloudlet server that includes the face recognition server IP address and port. Since the IP address is assigned by the Dynamic Host Configuration Protocol server because the VM is executing in bridged mode, however, the host server has no visibility into that assignment, so there was no simple way to obtain the IP address and port.

To solve this problem, we included a Windows service in the VM that runs on startup. The Windows service invokes a Python script that performs the following three tasks:

1. Start the face recognition server executable in a separate thread inside a Python script.
2. Read the face recognition server configuration file that contains the IP address and port that the face recognition server is listening on.
3. Write this information to a file that is accessible by the cloudlet.

Although the Windows service creates additional complexity on the cloudlet server, it reduces the complexity cloudlet setup in the field. During field operation, servers residing within the Tactical Operation Center and Humvees are provisioned with a set of prepackaged cloudlets to support a range of applications and versions to avoid provisioning servers for each supported application platform and version. The handheld devices of soldiers participating in the mission are then loaded with application overlays that are necessary for a particular mission. A soldier running a computation-expensive application can discover a compatible cloudlet within minutes and offload the expensive computation to the cloudlet running on a server.

What We've Learned

Our research has identified the following two types of applications that can be deployed in a cloudlet setting:

- *Data-source-reliant applications that rely on a particular data source to work.* For example, if soldiers need to launch the facial recognition application, they need a database of faces to match images with. In addition, if soldiers want to compare fingerprints, they need a database of fingerprints to match with. In this setting, the cloudlet must be configured to connect the cloudlet to a particular data source.
- *Non-data-source-reliant applications that are computationally intensive but don't require a large data source to work.* For example, imagine soldiers encountering a sign with characters they don't understand. They can take a picture of the sign and submit it to a cloudlet to determine the language in which the sign is written. In this case, the computationally-intensive code residing on the cloudlet relies on complex character recognition algorithms instead of a large database.

As expected, our experiments demonstrated that the size of the overlay increases overlay transmission time (which in turn consumes more battery) as well as VM synthesis time. If the data source is included inside the overlay, this would create a large overlay, which indicates that the cloudlet concept is better fit for non-data-source-reliant applications. We overcame this problem by specifying the location of the data source in a configuration file. The location could be the local server or a server accessible over a network or the Internet. Although this approach requires additional configuration, it is done only once (when the cloudlet is packaged by IT experts), rather than each time a server is configured in the field (potentially by non-IT experts).

Future Work

When testing the cloudlet prototype in the RTSS Concept Lab, we discovered that a reduced deployment time makes it easier to deploy an application in a tactical environment. We are working to capture those measurements and are developing the following applications to support our findings:

- **fingerprint recognition:** Fingerprints are captured using a fingerprint scanner connected to a handheld device and sent to the cloudlet for processing.
- **character recognition:** Pictures of a written sign are taken with a camera on the handheld device and sent to the cloudlet for character identification and translation.
- **speech recognition:** A voice speaking a foreign language is captured using the voice recorder on the handheld device and sent to the cloudlet for translation; the same application can be used to translate a response back to the identified foreign language.
- **model checking:** An app is generated on the handheld on-the-fly using end-user programming capabilities and sent to a model checker in a cloudlet to ensure it does not violate any security (or other) policies and constraints.

We will use these new applications to gather measurements related to bandwidth consumption of overlay transfer and VM synthesis to focus on optimization of cloudlet setup time.

Our future research and collaboration will position cloudlets to both reduce battery consumption and simplify application deployment in the field. For example, our goal is to use dynamic VM synthesis to slash the time needed to deploy applications, thereby shielding operators from unnecessary technical details, while also communicating and responding to mission-critical information at an accelerated operational tempo.

By Grace Lewis, Senior Member of the Technical Staff

Related Web Sites

http://blog.sei.cmu.edu/post.cfm/cloud-computing-for-the-battlefield

For General Information

For information about the SEI and its products and services, contact
Customer Relations
Phone: 412-268-5800
FAX: 412-268-6257
info@sei.cmu.edu
www.sei.cmu.edu

Research, Technology, and System Solutions

Group-Context-Aware Mobile Applications

Our modern data infrastructure has become very effective at getting the information you need, when you need it. It has become so effective that we rely on having instant access to information in many aspects of our lives. Unfortunately, there are still situations in which the data infrastructure cannot meet our needs due to various limitations at the *tactical edge*, which is a term used to describe hostile environments with limited resources, from war zones in Afghanistan to disaster relief in countries like Haiti and Japan. This article describes ongoing research at the SEI in edge-enabled tactical systems to address problems at the tactical edge.

At the tactical edge, the people who need the information most—warfighters, first responders, or other emergency personnel—depend on timely and valuable information to perform their tasks, or even to survive. Unfortunately, access to the information they need can be hard to achieve, for the following reasons:

- **information overload** stemming from too much information, coupled with an inability to locate truly vital information
- **information obscurity** due to a lack of awareness of the available information
- **resource scarcity** manifested as insufficient bandwidth, central processing unit (CPU) power, battery power, or even attention to get the needed information and continue to process, exploit, and disseminate it for as long as needed

We are tackling the information overload and information obscurity aspects of this problem by developing context-aware mobile applications.

A Different Approach to Context-Aware Mobile Applications

Context awareness in the mobile environment is not a new field of research. Most mobile devices come preloaded with applications that use location or time to account for user context. There is certainly no shortage of similar applications available for download. We decided, therefore, to explore alternative sources of data that would not only push the limit of what could be done with user context but also focus on the challenging environment at the tactical edge.

Our “eureka” moment came when we realized that when warfighters or first responders are at the tactical edge, they almost never operate alone. The most important contextual information to warfighters or first responders is the context of the people in the group, and how they relate to that context. This realization drove us to explore group context-aware mobile applications. These applications would, if built correctly, first consider individual user context and then relate that information to the group context, thereby helping users understand both their own states and the state of the group in which they participate.

Group context-aware mobile applications clearly have value at the tactical edge. For example, warfighters are well served by having access to positions of friends and foes on the battlefield. They could also benefit from supportive applications that monitor resources such as food, ammunition, or vital signs. With sufficient data and processing power, these applications could use historical trends to determine dynamically if a squad is walking into a possible ambush situation.

In other tactical environments, such as tsunami disaster areas, the ability to share information about resource needs, dangerous situations, or health emergencies in a structured way is also valuable. Such applications could tailor information to managers, construction workers, doctors, and other emergency personnel to help coordinate an effective emergency response.

Our research project, called Information Security to the Edge (ISE), explores the structure, applications, and implementation of a context model that includes group information. We have constructed a prototype application on the Android platform that implements the essential components needed by group context-aware mobile applications.

App Architecture: Logic and Data

The ISE prototype application follows the common model-view-controller (MVC) pattern, which decomposes an application into the following parts:

- **The model is the data.** This data is the information processed by the application. For example, the words typed by the user into a word processing application are data.

- **The view is the user interface.** For a word processing application, the view is the buttons, menus, scroll bars, and other visual effects provided by the application to help a user write a document.
- **The controller is the logic.** In the word processing application, the controller is the rules the application uses to save, present, filter, and otherwise modify the text. The function provided by each button or menu item can also be part of the controller.

Consistent with the MVC pattern, the ISE prototype has a central control mechanism that forms the “brains” of the application and manages data flow through it. The central controller coordinates data flow and processing through the following primary application elements:

- **The context engine** is the central processor for all context information used by the application. As device sensors report new data and applications on external devices send data to the local application, all data passes through the engine so that new events are detected as they occur. For example, if an external user sends GPS coordinates that indicate he is within 100 feet of a warfighter, then the device can alert the warfighter to his presence. Expanding on this concept, if a group task must be performed but everyone is working individually on other tasks, the local device can monitor task status and user position and report to the leader when all group members are ready and close by so the group task can be performed.
- **The sensor manager** accepts data from sensors that reside upon the mobile device. A typical smartphone contains position sensors, movement sensors, and in some cases, light and proximity sensors. The application captures data from these sensors and passes it through the sensor manager. The sensor manager enables the sensors and controls their sample rate, so the application can tailor usage to the situation and avoid overwhelming the system.
- **The communications manager** acts as the gateway to all external communications within the system. This gateway currently includes Bluetooth and TCP/IP communications but can include other communication mechanisms that are available to the device. Any messages to and from users on other devices are passed through the communications manager.

The sensor and communications manager architecture consolidates all sensor and communication concerns into a single location. This consolidation approach enabled us to build a standardized interface that simplifies integration of an arbitrary sensor (for example, a radiation sensor) or an arbitrary communication mechanism (for example, a line-of-sight radio that communicates with UAVs) with the application. We tested this feature through a collaboration with Joao Sousa of George Mason University. This testing resulted in the development of an alternative communication mechanism that integrates with the prototype with only a few weeks of effort, instead of months or years. We anticipate leveraging these standardized interfaces to collaborate with a variety of external groups and organizations as new sensor technologies and communication mechanisms become available.

App Architecture: User Interface (UI)

The ISE app, through the use of Android UI screens called *Activities*, reflects the view part of the MVC pattern. There are currently only three supported UIs in ISE:

- **User:** Allows users to look at the people with whom they are or can be connected, as well as the context data associated with each person.
- **Task View:** Allows users to create their own tasks, receive updates about other users’ tasks, and mark their tasks complete or incomplete.
- **Alerts View:** As events occur, some will automatically appear in the alerts view along with a list of the considerations the context engine has identified as items of importance for users. The alerts presented will be tailored to the needs and context of individual users.

We are upgrading the ISE architecture to support any UI that subscribes to standardized updates from the data services.

Challenges

One challenge we face involves accounting for the lack of network infrastructure. In particular, limited bandwidth exists for the available communication channels. We are building atop communication capacities that other organizations are field testing in Afghanistan to tailor our solution to practical field situations.

A second challenge involves providing warfighter access to backend data sources. Soldiers told us that important information is available in such sources, but they can’t readily find the relevant information. Moreover, they can’t access the database in the field. Other Advanced Mobile Systems work is investing ways to provide access to critical data through the use of cloudlets.

A third challenge involves reducing the user's cognitive load by limiting the amount of interaction and attention required of the user. Residents in a metropolitan area can use smartphones without undue concern with distraction, as long as they are not engaging in tasks that demand undivided attention. A soldier in Haiti, on the other hand, must be cognizant of crumbling buildings, while a warfighter on the ground in Afghanistan might need to digest information while taking enemy fire. Our goal is to use hardware that allows the warfighter to capture and process information seamlessly, without sacrificing valuable time and resources.

We are also addressing the challenge of resource scarcity. Resources are limited at the tactical edge and warfighters are typically limited to the power and bandwidth of whatever devices they can carry. We are therefore exploring resource optimization based on our expanded model of context. For example, if a warfighter's assignment involves driving through a known safe area, it may not be necessary for the smartphone to activate the GPS capability. By optimizing the system to use sensors only when needed, warfighters can save battery power, CPU cycles, and communication bandwidth that can be used to support other mission-critical needs.

Finally, our work will not have the desired impact if we cannot meet the challenge of relevance. Warfighters made it clear to us that if a device or application is not directly useful to their immediate task, it will be ignored. In any given day, a warfighter in Afghanistan may be asked to determine if a particular individual is a threat, sweep a village to establish identities of residents, deliver food to children, or check for a weapons cache. These different missions affect the type of information that interests soldiers and the type of information a software application should consider. Solving this problem requires a deep understanding of the needs of soldiers and the missions in which they engage. We are leveraging this domain knowledge so our ISE application can tailor information processing to a particular mission, thereby ensuring relevance to the current mission and the ability to change mission parameters as needed.

Looking Ahead

The ISE prototype is just one part of our strategy to address the problems of information overload, information obscurity, and resource scarcity. The Advanced Mobile Systems initiative is also engaged in other projects that address the three problems of information overload, information obscurity, and resource scarcity from different perspectives. We intend to integrate each project after they have matured, thereby providing an end-to-end solution to warfighters and first responders at the tactical edge.

By Marc Novakouski
Member of the Technical Staff

Related Web Sites

www.sei.cmu.edu/mobilecomputing
www.sei.cmu.edu/

For General Information

For information about the SEI and its products and services, contact Customer Relations
Phone: 412-268-5800
FAX: 412-268-6257
customer-relations@sei.cmu.edu
www.sei.cmu.edu