RK+MOSIX: A Real-time Kernel with Automatic Task Migration Support

http://www.ecs.syr.edu/faculty/oh/Research/RK+MOSIX/index.html

Jae C. Oh, Rajesh Chopade, Sunil Vajir, Radika Garg

1 Department of Electrical Engineering and Computer Science
Syracuse University, Syracuse, NY 13244
{jcoh|rchopade|svajir|rsgarg}@ecs.syr.edu

Abstract. We combine two well-known technologies, the resource kernel for real-time task guarantees and MOSIX for dynamic task migrations, and describe how to alleviate the problems that result from their interactions. We describe the ideas, motivations, and design and implementation of middleware based on these two separate systems. By combining a resource kernel (Linux/RK) and a dynamic load-balancing system (OpenMosix), we increase the chance of completing a real-time task, even if there aren’t enough resources in a local machine to meet the deadline of the task. Since the original Linux/RK was implemented for a single machine, not supporting task migrations, several issues relating to the resource kernel system calls had to be resolved before migration of real-time tasks was possible. We also resolved some compatibility issues in combining OpenMosix and Linux/RK. We discuss detailed implementation issues in combining these three systems and lessons learned. We also discuss network overhead issues relating to the admission control of real-time tasks.

Introduction

As the prices of computer components drop and the network connectivity increases, cluster and distributed computing paradigms become important research areas for computer scientists in both theory and practice. Utilization of abundant computing resources over a local-area network and beyond to the Internet is an attractive option to satisfy resource hungry applications.

We combine the ideas of the resource kernel [Oikawa and Rajkumar 1998] and dynamic load-balancing for distributed systems [Barak and La’adan 1998] to improve the timeliness of real-time tasks through a better utilization of distributed resources. We choose Linux/RK [Oikawa and Rajkumar 1998] and OpenMosix [Barak and La’adan 1998] for the combined system and we call the integrated system RK+MOSIX.

Linux/RK was developed for guaranteed completion of real-time tasks within respective deadlines when there are enough resources in a single computer. OpenMosix was developed for load-balancing in a cluster of computers through dynamic process migration. We chose this platform because Linux/RK is the only open source real-time operating system that is fully Linux compatible, and Mosix is the only open load balancing system with dynamic migration support that has stood the test of time.

Combining these two separate technologies requires us to deal with several theoretical as well as practical issues. For example, the theory has to be extended because
we need to design algorithms for remote admission control that will find an appropriate compute node for migration of a real-time task. Another example is taking into account the network overhead for obeying the timing constraints. As a practical matter, we need to resolve the issue of Linux/RK system calls for migrated processes, since Mosix routes some system calls back to the original node attempting to acquire resources in the original node after migration.

RK+Mosix is an interesting and promising project. It provides one of the first real systems that supports dynamic real-time task migration and remote admission control to improve timeliness of real-time tasks. The implementation of RK+Mosix is ongoing but we have completed the core part of the system that can provide significant insights into the design and development of real-time job migration middleware. Experimental studies for scalability and many other issues are currently considered but the results are not presented in this paper due to the space.

This paper is organized as follows. Section 2. describes Linux/RK and OpenMosix. Section 3. presents an overview of the RK+Mosix architecture. Section 4. discusses the steps taken in integrating Linux/RK and OpenMosix, issues encountered in the integration work, and their solutions. Section ?? presents the current status of the integrated system. Section 5. discusses future work. Finally, Section 6. concludes the paper and presents a summary.

Background

Linux/RK is a real-time kernel which provides timely, guaranteed, and protected access to system resources [Oikawa and Rajkumar 1998] under the Linux environment. Linux/RK allows applications to specify their own resource demands and the kernel satisfies those demands with some resource management schemes. A resource kernel can “convert” a general purpose operating system like Linux to an operating system which can be utilized in systems demanding guaranteed response time. To enhance portability, Linux/RK is developed as Linux kernel modules. More detailed information on RK can be found in [Oikawa and Rajkumar 1998].

OpenMosix is an extension of Linux kernel enabling adaptive load-balancing through process migration. Processes running on any node in the cluster can transparently migrate to another node where the CPU load is lower than the local node. The transparency support means that no special programming or linking to special libraries are required to take advantage of OpenMosix’s load-balancing technology.

OpenMosix has no central control; each node can operate as an autonomous system, and it makes all its control decisions independently. This design allows for dynamic configuration, where nodes may join or leave the network with minimal disruption. Additionally, this allows for a greater scalability and ensures that the system runs well on large configurations as it does on small configurations. Scalability is achieved by incorporating randomness in the system control algorithms, where each node bases its decisions on partial knowledge about the state of the other nodes, and does not even attempt to determine the overall state of the cluster or any particular node. For example, in the probabilistic information dissemination algorithm, each node sends, at a regular interval, defined by Mosix Frequency (MF), information about its available resources to a randomly chosen
subset of other nodes. At the same time it maintains a small window of list of node LoadList, with the most recently arrived information.

**RK+MOSIX Architecture**

Figure 1 shows an overview of the RK+MOSIX architecture explaining remote real-time task admission control and real-time task migration. When a real-time application with a deadline is introduced to the system (the local node in the figure), it is first submitted to the local RK admission control component. When the deadline cannot be honored by the resources in the local node, the RK admission control component will ask the local remote real-time task request daemon (RRTR daemon) for the possibility of a remote execution of the application. The RRTR daemon finds the most capable remote node by considering the load information of known remote nodes,\(^1\) and sends back to that node a real-time task request with the RK parameter list \{C, D, T\} for the application. The remote node’s Remote Admission Control (RAC) daemon then asks its own RK admission control component for the possibility of finishing the application on time. Then, the remote node’s RAC daemon transfers the answer from the admission control to the local RRTR daemon. If the control answers positively, the RRTR daemon requests the migration of the application to the local Mosix migration daemon (MIG daemon). Finally, the MIG daemon transfers the real-time application. When the remote node completes the migrated application, it will send necessary results back to the original node (i.e., the Unique Home Node (UHN)).

We facilitate the above mechanism by combining Linux/RK and OpenMosix.

\(^1\)Currently, we are considering only the CPU load information of remote nodes. We are investigating other factors for the capability of successful completion of real-time tasks.
Linux/RK consists of a set of RK modules that can be inserted after Linux/OpenMosix has booted up. Since Linux/RK and OpenMosix are not aware of each other, we had to resolve some of the issues involving Linux/RK system calls and OpenMosix process migration. We explain the issues and the current status of the system implementation below.

Integration of Linux/RK and OpenMosix

OpenMosix is an enhancement of Linux kernel through patch files. On the other hand, Linux/RK is implemented with kernel modules with minimal changes. The changes made in Linux kernel serve as “hooks” for RK modules. RK modules are inserted using the `insmod` command after the booting of Linux. There are four steps in integrating Linux/RK and MOSIX; each step is listed below and detailed in a subsequent subsection.

1. Upgrading Linux/RK to work with Linux version 2.4.20: There is kernel incompatibility between Linux/RK and OpenMosix as of the end of December 2003. We first upgraded the RK distribution to comply with the latest Mosix distribution.

2. Inserting Linux/RK kernel code to OpenMosix: We decided to insert Linux/RK code to OpenMosix because OpenMosix has many more files (about 200 files) than Linux/RK (about 25 files). Since Linux/RK and OpenMosix are not aware of each other, some conflicts are introduced after the insertion of the Linux/RK code. We explain in Section 4.2 those conflicts and how we have resolved them.

3. Supporting real-time task migration: After successfully inserting Linux/RK code to OpenMosix, we designed and implemented real-time task migration so that migrated real-time tasks could run on remote machines.

4. Supporting automatic real-task migration: The last step is to design and implement a component that communicates with remote nodes and automatically identifies a suitable remote node for a successful real-time task completion after migrating to the node.

We explain details of the integration steps in the following.

Upgrading Linux/RK to work with Linux version 2.4.20

When we started to develop RK+MOSIX in December 2003, the Linux/RK distribution worked for the Linux kernel 2.2.14. Unfortunately, the most recent version of OpenMosix was for the version 2.4.20. We’ve decided to upgrade Linux/RK for the version 2.4.20. The decision was made partly because Linux/RK comes with about twenty five source files where as OpenMosix comes with about two hundred source files; it was relatively easier to upgrade Linux/RK to work with Linux 2.4.20. The entire upgrading took a little over a month. The RK source code is distributed as an entire Linux source tree rather than as kernel patch files. This made the upgrade and integration work more cumbersome. Since OpenMosix is distributed as kernel patch files, we’ve decided to make new patch files for RK+Linux. We extracted the Linux/RK kernel code from the RK-enabled Linux tree first. Then, we’ve created a `diff` file against the pure Linux 2.4.14 to create a sensible patch file. Finally, we manually patched the required code from this file to OpenMosix for the Linux version 2.4.20 to create a small manageable patch file for RK+Mosix. Now we can simply use the Unix patch command to embed RK to Linux.

Another problem we’ve encountered in this step was related to the high precision
timer used by Linux/RK. Under the 2.4.20 kernel, the high precision timer didn’t work due to conflicts with the SMP support in the kernel. The high precision timer in RK works by configuring CMOS 8254 timer. We disabled the SMP feature to make the timer working again. The problem seems to arise because of the conflict with APIC timers that is enabled with the SMP support on Linux 2.4.20.

**Inserting Linux/RK kernel code to OpenMosix**

After making Linux/RK work with Linux 2.2.20, the next step was to merge Linux/RK and OpenMosix. There were many conflicts because some files were relevant to both Linux/RK and OpenMosix. Thus, simply using the kernel patch created in the above step did not work. The problem is that when we applied the new Linux/RK patch files that are also modified by OpenMosix, the patch would not work properly; we went through these files and give preference to OpenMosix or Linux/RK depending on the situations. In the end, we were able to produce a compilable single image of the first version of RK+Mosix.

**Reconciling Linux, Linux/RK, and OpenMosix system calls**

The single image created in the previous step supported the features of OpenMosix well (i.e., normal process migration) but the Linux/RK features were not properly supported. The Linux/RK specific system calls were not recognized because most of the functionality of Linux/RK was implemented with kernel modules rather than kernel patches. In other words, Linux/RK modules are inserted using the `insmod` command after booting. OpenMosix, on the other hand, was implemented as series of kernel patches making OpenMosix part of Linux kernel.

As OpenMosix boots, it reads the system call information from the Linux system call table and creates its own system call table: the *remote system call table*. For a migrated process, certain system calls are satisfied locally in the remote node while other system calls such as file I/Os may need to be served by the UHN. OpenMosix handles this requirement with the remote system call table. At the time of boot, since Linux/RK modules have not been inserted yet, OpenMosix misses the opportunity to build entries for the Linux/RK system calls in the remote system call table.

Figure 2 illustrates how we resolved the problem of OpenMosix not recognizing Linux/RK system calls. We added Linux/RK system call entries in the Linux system call table (depicted as *Linux Local System Call Table with RK syscall in the figure.*) but the system call pointers point to dummy functions that just return “Linux/RK OK.” When OpenMosix boots up (but Linux/RK modules are not inserted yet), it will create the local system call table with entries for the dummy Linux/RK system call entries. After booting, when Linux/RK modules are inserted, the dummy pointers are replaced with pointers to the appropriate RK syscall handlers. Notice that we could also have directly updated OpenMosix’s remote system call table to point to the RK system call.

---

2From this point on, when we say “OpenMosix system call table,” we mean “OpenMosix remote system call table.”

3The name “remote system call table” is coined because the table is used after a process has migrated to a new node.
handlers. However, we wanted to limit the changes in the RK code as much as possible.

When RK+OpenMosix makes a Linux/RK system call, it will first find it in the
OpenMosix system call table then the call is directed to the corresponding Linux/RK
entry in the Linux system call table. The Linux/RK entries now have the correct pointers
to the proper Linux/RK system call handler routines.

Supporting automatic real-time task migration

After resolving the system call issue, we addressed migration of real-time tasks
from a node to another. When a real-time task is created, a resource set (RS) and the CPU
reserve for the task are created. The CPU reserve is then attached to the RS. Finally, the
real-time task is attached to the RS.

A CPU reserve in Linux/RK uses a high resolution timer for scheduling. When
an attempt is made to create a CPU reserve for a migrated task, the home node’s high
resolution timer refuses to honor the “create CPU reserve” system call. This is not ex-
actly a problem since the CPU reserve should be made in the local node where the task
migrated. However, in general, the problem is that when a migrated real-time task makes
a Linux/RK system call, the system call is routed back to the UHN of the migrated task
instead of being served by the local kernel. This is because after a task has been migrated,
a flag is set to indicate the task is a migrated task. When a system call is made by the task,
the system call is honored by the information provided by the remote system call table of
OpenMosix.

Since we want to see most of the real-time scheduling performed by
the local node, we modified the way system calls are handled. The file
arch/i386/kernel/entry.S that handles system calls is configured in such a way that if the system call number is higher than a certain threshold (we chose 256 for this implementation), the system call is honored by the local system call table.

In fact, this solution made the modification described in Section 4.3. unnecessary because now Linux/RK system calls are directly served by the local system call table. However, we have decided to keep the modification since keeping OpenMosix aware of Linux/RK system calls is a desired outcome for future development of RK+Mosix.

For deciding which node to migrate the task, we introduced a new system call, \texttt{rk_admit_cpu_query()}, which is used to query remote nodes for availability of the resources. If the current node is not able to support the job deadlines, the query is directed to other nodes in the cluster. We utilize the \texttt{consider()} function provided by Mosix to send the real-time information (i.e, C, D, and T) associated with the job. On the receiving end, the info daemon checks if the information received is Mosix information or RK information. If it is RK information than query is made to its own node with these RK parameters. Then the node replies if it is able to accept the job. If the remote node is able to accept the job, then the job is marked for migration. A call is made to the scheduler of the home node. If scheduler sees a job is marked for migration, it initiates migration to the remote node. All this is executed within the system call and transparent to the user. The system call returns normally when any of the nodes is able to accept the job and fails if none of the job accepts the job.

\textbf{Ongoing Efforts and Discussions}

In migration of a real-time task, network overhead must be considered. There are two categories of network overhead in running a real-time task on a remote node: (1) \textit{migration overhead} and (2) \textit{run-time overhead}.

We can predict the migration overhead fairly accurately by considering the number of pages to be migrated.

The run-time overhead is more difficult to predict. The run-time overhead is incurred mostly by making system calls to the UHN from the remote node. Since the number of system calls to be made and those system call types are not statically known, it is hard to predict the overhead. What we have to do is to reserve a certain amount of network bandwidth for the migrated process so that the system calls routed to the UHN are serviced in a timely manner. Experiments can be designed to test the network overhead. For example, we can set up two computers with a known network speed connection. We need to measure how long it takes to serve each system call routed back to the UHN. Another possibility is to consider how I/O is handled for migrated tasks instead of always making I/O calls back to the UHN.

Ghosh \textit{et al} [Ghosh and Rajkumar 2002] discusses network resource management support for Linux/RK. The solution provides direct control over the network packet flows, guaranteed and enforced processing time for the received packets, and precise accounting of the packet processing times.

We are also investigating the feasibility of using the netnice [Okumura et al. 2000] for network bandwidth reservation. The netnice project started
with a network bandwidth control mechanism for user processes [Okumura et al. 2000].

Real-time tasks also require reservation of disk resource. In most of cases, the disk I/Os must be reserved in the UHN even after a task has been migrated. We will study the trade-off between migration of data and remote disk access.

Currently, once a real-time task has started, it doesn’t get preempted during the execution. We are investigating the feasibility of interrupting a task and consider it for migration to accommodate more urgent task’s deadline.

Lastly, after a migration, the accepting node’s load may have increased preventing timeliness execution of the migrated job. In this case, the simplest solution may be migrating the job again. There are many problems related to this but we will discuss this in future papers due to the space limitations.

Summary and Current Status

We have described a design and implementation of integrating two well-known technologies: Linux/RK and OpenMosix. There were several technical obstacles to overcome the integration of these two systems such as conflicting system calls, remote system calls, and task migration, among others.

The current system can migrate real-time tasks to the least heavily loaded node known, reserve required CPU resource in the remote node, and complete the task successfully within the specified deadlines. We have tested the system with over ten different user programs in various sizes without a problem found. The test programs invoke all Linux/RK system calls, most of Linux system calls, and all OpenMosix calls. It took about six months of work in developing the current version. Currently, we are studying various algorithms for choosing the target node for migration.

RK+Mosix shows a great potential for supporting real-time tasks in distributed systems and cluster computing environments.

Acknowledgments

This work was partially funded by NSF/ITR GRANT: # 0121658.

References


