A Framework for Middleware Supporting Real-Time Wide-Area Distributed Computing

K. H. (Kane) Kim, Stephen Jenks, Larry Smarr, Andrew Chien, and Liang-Chen Zheng

Electrical Eng. and Computer Science Dept.
University of California, Irvine
{khkim@uci.edu, sjenks@uci.edu, lzheng@uci.edu}

Dept. of Computer Science and Engineering
University of California, San Diego
{lsmarr@ucsd.edu, achien@cs.ucsd.edu}

Abstract: *The development of real-time systems that are distributed over a wide area is a significant problem that has challenged researchers for many years. Such systems require precise timing of actions throughout, but wide area networks tend to have variable delays that greatly reduce determinism.*

The framework described here supports the composition of wide-area real-time Distributed Virtual Computer systems from deterministic components to provide precise timing from end-to-end. The framework will use switched optical networks such as those being developed for the OptIPuter project for well-regulated long-distance paths yielding low jitter. Within campus networks, the Time-Triggered Ethernet technology will provide deterministic switching of packets for real-time applications while supporting legacy IP traffic. Finally, at the edges, the Time-triggered Message-triggered Object (TMO) Support Middleware will manage resources to provide real-time operations while supporting an easy-to-program interface and high-level timing specification.

1. Introduction

The history of distributed computing (DC) spans more than three decades but the branch of the DC technology related to real-time (RT) applications still remains in an immature state. The needs for correcting this situation are becoming more acute than ever because exploration of new-generation DC applications which involve *actions subject to relatively high-precision timing requirements* has been exploding in recent years. Multi-media network based applications, collaborating robots, and sensor network based applications are representative of the exploding RT DC application fields.

At the top level the biggest fundamental research challenge is:

To enable high-level high-precision real-time distributed software design and programming accompanied by cost-effective guaranteeing of acceptable response times.

High-level RT DC programming is a research subject of critical importance in this decade since a vast number of new-generation RT DC application software cannot be constructed in easily analyzable and thus highly reliable forms with acceptable costs by use of conventional C or assembly-level programming. Technologies for cost-effective guaranteeing of acceptable response times are also of great importance. If response time requirements are not met, then obviously applications fail. The efforts for guaranteeing acceptable response times must also be affordable. The state of the art in this area is very weak.

The top-level goal which the practicing community has been longing for over many years cannot be realized without successful development of some key component technologies such as (1) programming model and API; and (2) middleware possessing effective resource management capabilities.

The essence of RT computing is to *effect important output actions within precisely specified time-windows*. In RT DC systems, if an output action of a node becomes ready only after manipulating some data coming from another node, then the production of the data by the latter node and its communication to the former node must all occur *in time* for enabling the output action to be within the specified time-window. Controlling communication delays is obviously much easier in LAN (local area network) -based application systems than in WAN (wide area network) -based systems. Naturally, research in LAN-based RT DC systems has advanced steadily over the past two decades, but fewer research efforts have been made in the field of WAN-based high-reliability RT DC systems.

Recently, efforts for creating WAN environments in which bounds on communication delay jitters are significantly smaller than those in most major segments of the current Internet started growing. For example, efforts for creating optical grids yielding tightly bounded delay jitters are under way [Sma03]. Therefore, we expect that research on RT wide-area DC will be accelerated from here on.

At the more basic level, WAN-based DC has become an active field of research, especially under the label of *grid computing* [Fos01]. As the grid computing research community grows, the branch of the community which deals seriously with the quality-of-service (QoS) aspect is showing increasing interests in the potential for wide-area RT DC. We have defined an *RT grid* (or an *RT sub-grid*) as:

a grid (or a sub-grid) which facilitates both (RG1) message communications with *easily determinable tight latency bounds* and
(RG2) computing node operations enabling *easy guaranteeing of timely progress of threads toward computational milestones* [Kim04].

What has prevented the creation of a broad range of RT grids? The issue of transferring messages with easily determinable tight latency bounds has been the main stumbling-block, particularly when coast-to-coast latencies are greater than ten milliseconds and world-wide latencies are greater. These latencies are often greater than the period of real-time tasks at either end of the connection. Therefore, scheduling of the communications and computations requires consideration of the number of messages in flight and the communication delay.

The issue of jitter and indeterminate delays along these long distance links is largely one of economic constraints. Dedicated paths were unaffordable for most applications. The situation is changing, however. Continuous increase in the availability of wavelengths of light, called *Lambdas*, traversing strands of optical fiber, combined with the steady declines of the costs of acquiring such lambdas, have encouraged some technological visionaries to propose to use dedicated Lambdas to form fully optical data paths among major science laboratories spread over multiple cities and states. A challenge then is how to extend and adapt the most advanced and effective technologies established for RT DC in LAN environments to such RT grid environments. Although a fundamental obstacle may be absent in such dedicated optical-path WAN environments, there are many research questions related to achievable DC performance, optimal use of resources, newly enabled applications, etc.

In fact, campus networks, including many university campus networks and networks covering industrial complexes, typically involve multiple Ethernet switches. Therefore, RT message communication through this LAN part of the RT grids is a non-trivial issue. However, there has been a breakthrough produced by Hermann Kopetz at the Vienna Univ of Technology (TUW). The new RT LAN scheme invented by him is called the *Time-Triggered (TT) Ethernet* [Kop05].

2. Real-Time Distributed Virtual Computers

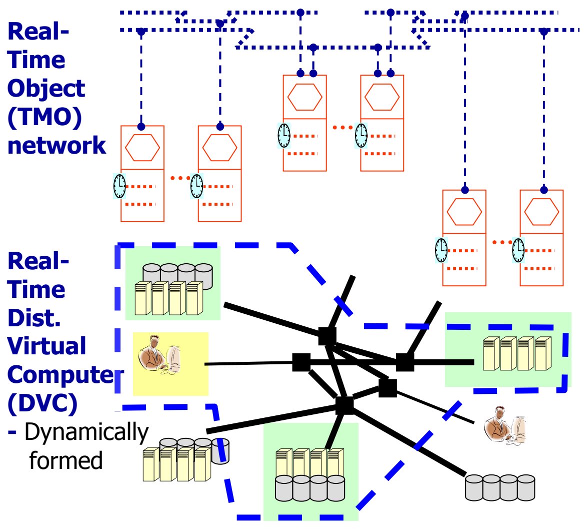
Real-Time Object (TMO) network
Real-Time Dist. Virtual Computer (DVC)
- Dynamically formed

Figure 1. A real-time DVC

Once an RT grid with appropriate RT communication capabilities is established, it can be operated to provide a different set of *RT distributed virtual computers* (DVCs), also called RT DC virtual machines, at a different time. The term DVC was used by Larry Smarr and Andrew Chien to refer to the configuration of DC resources with the following characteristics [Sma03, Kim04] and depicted in Figure 1.

(DVC1) A DVC is an image of a DC facility presented to application software. It can be viewed as a *dynamically established sub-grid*. It typically involves computing nodes, which may come from one site or multiple sites, and communication paths among the nodes. A computing node here may contain processors, storage devices, displays, other peripherals, and ports leading to inter-node communication paths.

(DVC2) A DVC is configured dynamically in principle. However, it may remain in tact for a long time, e.g., even for months, or for as short as a few minutes. Within a DVC, multiple applications may run concurrently, sharing resources of the DVC.

Ideally, a DVC should look like "a safe local cluster" to the applications, rather than like the "relatively hostile, best-effort, open Internet" [Sma03]. However, the relatively long message communication latency inherent in WANs puts a limit on the extent to which a DVC can be made to look like a local cluster, at least to RT application developers.

Based this DVC notion, we have established its specialized case for RT computing, i.e., the notion of RT DVC. An RT DVC must be distinctively stronger than a general DVC in the following characteristics [Kim04]:

(RD1) The message paths in an RT DVC must have the capability of transferring messages with easily determinable tight latency bounds. Therefore, if a lambda-path were to support such *RT message paths* for one RT DVC together with message paths for other simultaneously-active DVCs, it must be operated to provide both tight latency bounds and guaranteed bandwidths to the RT DVC.

(RD2) In each computing or sensing-actuating site (offering computing nodes) within the RT DVC, participating computing nodes must exhibit timing behaviors which are not different from those of computing nodes in an isolated site by more than a few percents. Also, computing nodes in an RT DVC must enable easy procedures for assuring the very high probability of application processes and threads reaching important milestones on time. This means that computing nodes must be equipped with appropriate infrastructure software, i.e., OS kernel and middleware with relatively easily analyzable QoS.

(RD3) If representative computing nodes of two RT DVCs are connected via RT message paths, then the ensemble consisting of the two DVCs and the RT message paths is also an RT DVC. This makes the definition of RT DVC to be recursive in nature.

An RT DVC may be dedicated to running one RT DC application or used to support multiple RT DC applications concurrently. Therefore, extensive research is needed to establish a middleware model with the following capabilities:

(M1) Establishment of RT communication paths spanning wide areas, including allocation of Lambdas, allocation of *virtual Lambdas* (i.e., time-shares of Lambdas), and coordination of RT message-send timings; A middleware subsystem handling these may be called an *RT communication infrastructure management* (RCIM) subsystem.

(M2) On-demand creation of RT DVCs; A middleware subsystem handling these may be called an *RT grid resource management* (RGRM) subsystem. The RGRM subsystem may learn from the RCIM subsystems what RT communication paths are available for use. It allocates a subset of the available computing resources (including PC clusters in DC sites) and communication resources to the new RT DVC being created in response to a request.

(M3) Allocation of both computing and communication resources within the DVC as demands from the applications arise; A middleware subsystem handling these may be called an *intra-RT-DVC resource management* (IRDRM) subsystem. It seems reasonable to attempt to develop an IRDRM subsystem as an extension of a well established middleware subsystem effective in LAN-based RT DC systems.

In this paper, we present a middleware framework consisting of the three middleware subsystems mentioned above and discuss some major issues that need to be resolved to realize the middleware in a cost-effective form.

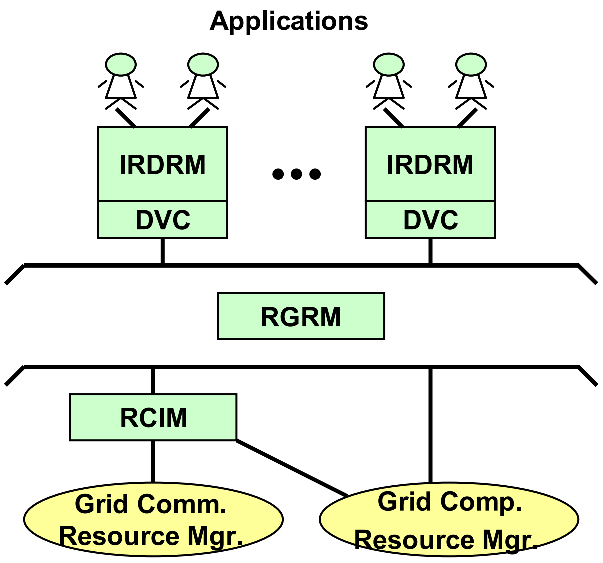

Figure 2. Structure of the RT DVC middleware

3. RT DVC Middleware

The structure of the RT DVC middleware is depicted in Figure 2. The RCIM subsystem obtains appropriate resources from both the general grid computing resource manager and the general grid communication resource manager and then establish RT communication paths spanning wide areas. The RGRM subsystem obtains appropriate computing resources from the general grid computing resource manager and obtains RT communication paths from the RCIM subsystem and use them to configure one or more RT DVCs as demands arise. On each RT DVC, the IRDRM subsystem supports multiple RT wide-area DC applications.

Major issues and approaches in realizing each middleware subsystem are discussed below.

3.1 Technical foundation for realizing high-QoS WAN communication layers

As we defined, an RT grid must facilitate message communications with *easily determinable tight latency bounds*. The WAN in which an RT grid resides contains typically three different regions as depicted in Figure 3. For Region 2 and Region 3, a common approach for creating an environment yielding tight latency bounds has been to dedicate the LAN to the communication among the constituencies of RT DC in the local area and isolate it from non-RT computing sites in the local area and from the general Internet. The economic incentives for using more flexible networking schemes than such isolated LAN schemes are strong. Recently, a technology breakthrough that allows more flexible local area networking without impairing the tight latency bounds came from Hermann Kopetz at TUW [Kop095]. The basic idea behind the newly invented RT LAN scheme, *TT Ethernet*, is to put a special mark in the message header that indicates whether the message is an RT message or a normal message. Whenever an RT message arrives at a switch, the message is processed first even if some non-RT messages are already in the buffer within the switch.

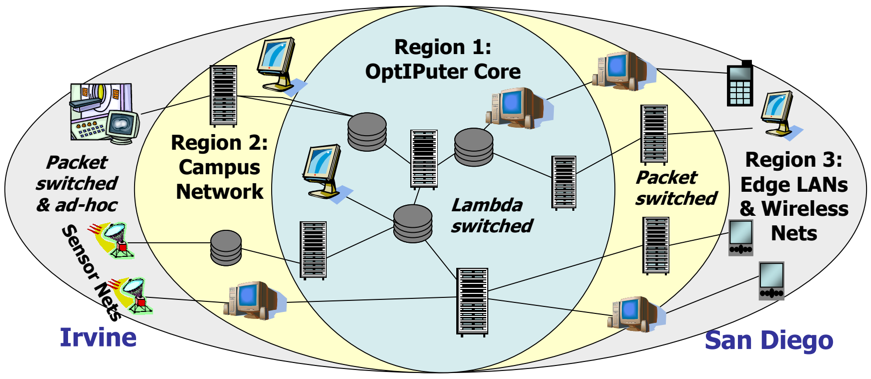

Figure 3. Three regions in a WAN

The generators of RT messages must cooperate so that the RT messages produced by them may traverse in collision-free manners. This cooperation is handled by instantiations of the middleware running on DC sites and development of such middleware is an important research topic. We have some experiences in building special instances of such middleware during the course of developing TMOSM (*TMO Support Middleware*) [Kim99]. However, the cooperation logic developed there is a simple TDMA support logic which covers simple Ethernet configurations but does not cover complex campus network configurations involving tens of Ethernet switches. Therefore, establishment of a middleware subsystem which schedules / coordinates RT message-send actions of DC sites in manners effective in typical campus network environments equipped with TT Ethernet switches is an open research issue.

For Region 1, a new-generation WAN backbone, in particular, the OptIPuter core being developed under the leadership of Larry Smarr at UCSD, is an attractive choice. Three possible situations must be considered in developing an RCIM subsystem dealing with such a WAN backbone. In one case, Lambdas can be reserved for various lengths of time as a part of forming dedicated message paths between gateways in Region 2. In this case, the network communication jitter and throughput variance are negligible and thus such Region 1 presents no more problems in realizing high-QoS WAN communication layers. In the second case, a time-share of a Lambda can be reserved such that a tight latency bound can be easily determined and a certain bandwidth is guaranteed to an RT message path from one gateway to another in Region 2. Therefore, certain types of high-QoS WAN communication layers can still be realized in this case. In the third case, Lambdas cannot be reserved for dedicated use or guaranteed time-shares but it is possible to program Lambda switches to support RT messages in the manner that the TT Ethernet switch does. This will require the cooperation of the vendor of the Lambda switches. We plan to study all three cases in the context of the special connections being established between UCI and UCSD partly to enable realization of a Southern California OptIPuter.

Effective integration of the middleware subsystem facilitating harmonious communication of RT messages through Region 2 and Region 3 with the subsystem managing RT message paths in Region 1 is also an important research topic. The integrated RCIM subsystem can then handle establishment of RT communication paths spanning wide areas, including allocation of Lambdas, allocation of virtual Lambdas (i.e., time-shares of Lambdas), and coordination of RT message-send timings.

3.2 An RT grid resource management (RGRM) middleware subsystem handling on-demand creation of RT DVCs

The RGRM subsystem responds to requests from users for creation of RT DVCs. It must first be able to keep an RT subgrid by obtaining computing node facilities, RT communication paths, and peripherals from the grid. RT communication paths can be obtained from the RCIM subsystem while computing nodes can be obtained from the general grid computing resource manager.

The RGRM subsystem allocates a subset of the available computing resources, including PC clusters in DC sites, and communication resources to the new RT DVC being created in response to a request. When the lifetime of the RT DVC is over, all the resources used to form the RT DVC are returned to the RGRM subsystem. Since the costs of computing node facilities are still declining at a steady rate, it seems reasonable to consider only dedicated use of computing nodes in each RT DVC. However, significant segments of each RT communication path available in the RT grid may be virtual links or path-segments to begin with. Therefore, the allocation of RT communication paths by RGRM to an RT DVC may involve splitting of a virtual path into multiple virtual paths of narrower bandwidths and increased delay bounds. As a systematic splitting technique is needed in the RCIM subsystem, a similar technique is needed in this RGRM subsystem as well.

The Grid Resource Allocation and Management (GRAM) service, a part of the Globus toolkit [Fos97, Fos98], will need to be extended to support RGRM. It has been pretty much CPU-usage oriented, with RSL (Resource Specification Language) attributes like *hostCount*, *maxCpuTime*, *maxMemory*, etc. RT DVC is additionally concerned with RT communication paths. Attributes for such connectivity with associated latency and bandwidth characteristics will need to be added to GRAM and RSL.

3.3 An intra-RT-DVC resource management (IRDRM) middleware subsystem and an RT DVC programming model

The IRDRM subsystem handles allocation of both computing and communication resources within the DVC as demands from the applications arise. The ultimate goal of the resource acquisition of an RT DC application is to execute its logic in time for timely execution of output actions. Past research on resource management in RT DC systems has taught us two important lessons:

(CI1) Many DC actions occur concurrently and involve use of both computing and communication resources. Past research dealing with computing resource allocation was extensive but largely conducted in separation from the research on allocation of communication resources. An exceptional case is the work by the Kopetz' group although the application programming model adopted there is of a somewhat restrictive type [Kop97]. To maximize the computational efficiency of many concurrent RT DC actions, a well orchestrated allocation of both computing resources and communication resources is desirable and is an important research challenge in this decade.

(CI2) Efficient resource management in RT DC systems is not possible without reasonably accurate *derivation of resource demands from the specifications of RT DC applications*, i.e., RT DC application programs. Accurate derivation of resource demands from RT DC programs is very difficult with RT DC programs specified in low-level programming styles, i.e., the styles practiced in C or assembly languages.

In the past decade, much of our research was directed toward establishing a high-level RT DC object model, the *Time-triggered Message-triggered Object* (TMO), which includes parameters that can be easily interpreted by the resource manager (i.e., the node kernel and the resource management middleware subsystem) and lead to efficient resource management. The main goal has been to relieve the RT DC application designers and programmers of the burden of dealing with low-level programming tools and low-level abstractions of computing and communication environments [Kim97, Kim00, Kim02a].

During the initiation of the TMO model, we thought that desired RT DC object models must include timing specification mechanisms which were intuitive and simple to use and yet possessed strong expressive power. Such timing specifications must enable relatively easy interpretation by the DC resource management middleware subsystem. The TMO model reflects our judgment that supporting programmers to specify *start-time-windows* and *completion deadlines* of *RT computation-segments* in convenient manners, rather than simplistically specifying priority numbers, is very important. After all, it is natural for RT application programmers to think about start-time-windows and completion deadlines rather than priority numbers.

In addition, it is desirable to support programmers to specify a *deadline for result arrival* in association with a remote method call. It will enable systematic composition of higher-level services with timeliness assurances out of lower-level services.

We have been enabling TMO programming without creating any new language or compiler. Instead, a middleware model called the TMOSM (*TMO Support Middleware*) provides execution support mechanisms and can be easily adapted to a variety of commercial kernel+hardware platforms compliant with industry standards [Kim99]. TMOSM uses well-established services of commercial OSs, e.g., process and thread support services, short-term scheduling services, and low-level communication protocols, in a manner transparent to the application programmer. The TMOSM architecture was devised to contribute to simplifying the analysis of the execution time behavior of application TMOs running on TMOSM.

TMOSM has been found to be easily adaptable to most commercial hardware + kernel platforms, e.g., PCs or similar hardware with Windows XP, Windows CE, Linux, etc. [http://dream.eng.uci.edu/ TMO-download/]. Our experiences indicate that even this middleware extension of a general-purpose OS such as Windows XP can support application actions with the 10ms-level timing accuracy. A Windows CE based prototype of TMOSM has been developed and is under continuous optimization with the goal of supporting application actions with better-than-10ms-level timing accuracy. Multiple prototype implementations of TMOSM and similar middleware based on the Linux platform also exist [KimH02].

A friendly programming interface wrapping the execution support services of TMOSM has also been developed and named the *TMO Support Library* (TMOSL) [Kim00]. It consists of a number of C++ classes and approximates a programming language directly supporting TMO as a basic building-block. The programming scheme and supporting tools have been used in a broad range of basic research and application prototyping projects in a number of research organizations and also used in an undergraduate course on RT DC programming at UCI [http://dream.eng.uci.edu/eecs123/serious.htm]. TMO facilitates a highly abstract programming style without compromising the degree of control over timing precisions of important actions. However, research on the TMO scheme has been largely confined to the cases of RT local-area DC.

It seems reasonable to attempt to develop an IRDRM subsystem as an extension of a well established middleware subsystem effective in RT local-area DC systems. We plan to extend the TMO programming scheme into a desirable programming model for use in RT wide-area DC application programming. A desirable programming model should allow natural intuitive specification of not only the timeliness and other QoS requirements inherent in RT local-area DC applications but also those in RT wide-area DC applications. Similarly, we believe that TMOSM is a solid starting base for developing a desirable IRDRM middleware subsystem model. Rationales for these adoptions are elaborated in the sections to follow.

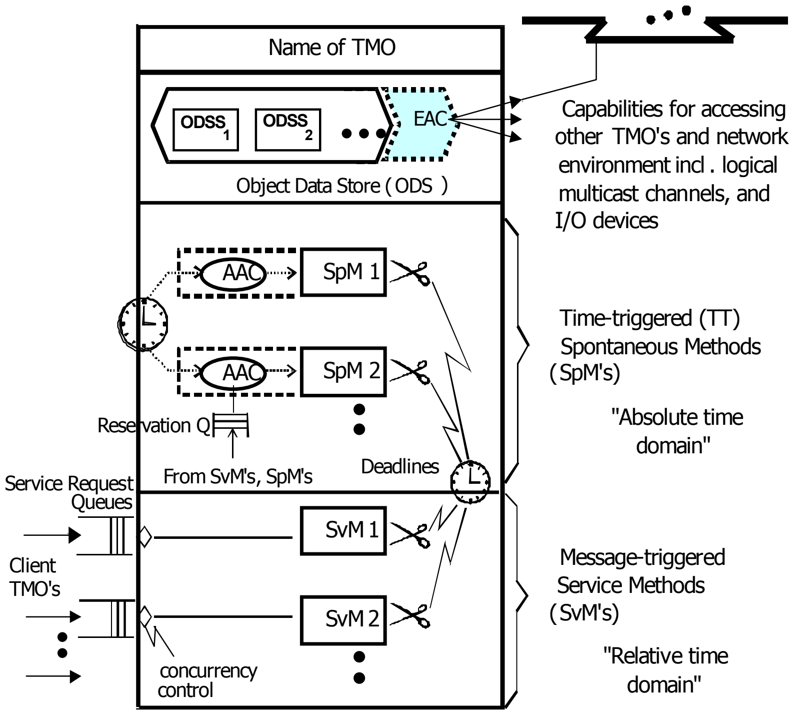

Figure 4. TMO structure

4 . The TMO scheme as a starting point for establishment of an RT DVC programming model

The key features of the TMO programming and specification scheme are reviewed.

(TM1) All time references in a TMO are references to *global time* in that their meaning and correctness are unaffected by the location of the TMO [Kop97]. If GPS receivers are incorporated into the TMO execution engine, then a global time base of microsecond-level precision can be established. Within a cluster computer a master-slave scheme, which involves time announcements by the master and exploitation of the knowledge on the message delay between the master and the slave, can be used to establish a global time base of sub-millisecond level precision.

(TM2) TMO has been devised to contain only high-level intuitive and yet precise expressions of timing requirements. *Start-time-windows* and *completion deadlines* for object methods and *time-windows for output actions* are used but no specification in indirect terms (e.g., priority) are required. *Deadlines for result arrivals* can also be specified in the client's calls for service methods.

(TM3) TMO is a high-level program construct in that conventional low-level program abstractions such as processes, threads, priorities, and socket communication protocols are transparent to TMO programmers. Yet, it offers a powerful structure which is capable of representing all conceivable practical RT and non-RT applications in easy-to-analyze forms.

(TM4) TMO is a natural, syntactically minor, and semantically powerful extension of the conventional object(s). The basic structure is depicted in Figure 4. TMO is a DC component and thus TMOs distributed over multiple nodes may interact via *remote method calls* and another mechanism (see TM6).

(TM5) TMO is also an *autonomous active* DC component. Its autonomous action capability stems from one of its unique parts, called the *time-triggered (TT) methods* or the *spontaneous methods* (SpMs), which are clearly separated from the conventional *service methods* (SvMs). The SpM executions are triggered upon reaching of the real-time clock at specific values determined at the design time whereas the SvM executions are triggered by service request messages from clients. For example, the triggering times may be specified as "for t = from 10am to 10:50am every 30min start-during (t, t+5min) finish-by t+10min". By using SpMs, *global time based coordination of distributed actions* (TCoDA), a principle pioneered by our international collaborator Hermann Kopetz [Kop97], can be easily designed and realized.

(TM6) TMOs can use another interaction mode in which messages can be exchanged over logical multicast channels of which access gates are explicitly specified as data members of involved TMOs. The channel facility is called the *Real-time Multicast and Memory-replication Channel* (RMMC) [Kim00]. The RMMC scheme facilitates RT publish-subscribe channels in a powerful form. It supports not only conventional *event messages* but also *state messages* based on distributed replicated memory semantics [Kop97].

(TM7) The TMO incorporates several rules for execution of its components that make the analysis of the worst-case time behavior of TMOs to be systematic and relatively easy while not reducing the programming power in any way [Kim97, Kim00].

(TM8) An underlying design philosophy of the TMO scheme is that an RT computer system will always take the form of a network of TMOs, which may be produced in a top-down multi-step fashion [Kim97]. For example, the earthquake monitoring and coordinated response application can be implemented in the form of a TMO network as depicted in Figure 5. Each earthquake monitoring station sends its pre-analyzed sensing reports to one or more earthquake response control centers (ERCCs). Each ERCC analyzes the received sensing reports and makes a judgment on the needs for issuing alert notices to various sensitive factories, e.g., nuclear power plants, hospitals with surgery rooms, semiconductor manufacturing plants, biological experimentation laboratories, etc. If proper alert notices arrive in time, then each receiving factory can take actions aimed for averting or minimizing the damages to its facilities. The timeliness of the actions of all components involved in this scenario is of crucial importance. Without highly reliable RT DVCs, this kind of applications cannot be realized.

The attractive nature of the TMO scheme as a starting point for establishment of an RT DVC programming model, can be summarized as follows:

(TA1) The high-level nature of the scheme makes application programming with the RT DVC easier than the programming in other lower-level styles.

(TA2) The extremely broad application coverage of the TMO network programming scheme enable researchers dealing with issues such as RT DVC middleware, optimal executions of RT DVC application software, and so on to focus on handling all possible types of TMO instances. Once they have considered all possible TMO instances, then they know that they have in effect considered all possible practical situations of RT DVC applications.

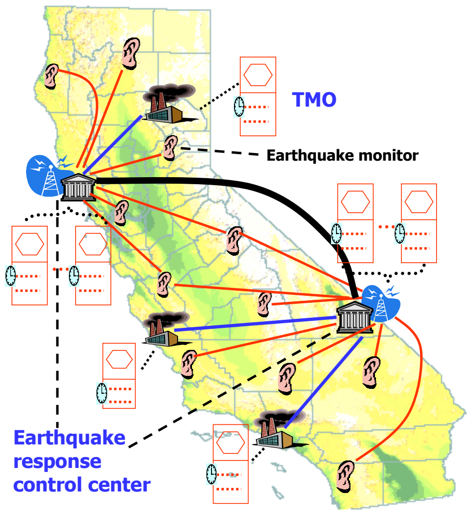

Figure 5. A network of RT objects handling earthquake monitoring and coordinated response

(TA3) The use of global time and convenient support for TCoDA provided by the TMO scheme enable both RT DVC middleware designers and RT DVC application software designers to exploit TCoDA for optimizing the performance of their products. The large communication latency inherent in an RT DVC occupying a large geographical region makes it compelling to exploit TCoDA. Compared to conventional hand-shaking message-based coordination of DC actions, this TCoDA approach can be multiple times more efficient and reliable. For example, three endpoints can be designed to simultaneously start at 10AM to take certain coordinated courses of actions without exchanging any synchronization messages if they observe certain conditions by 9:59AM.

5. TMOSM as a starting point for establishment of an IRDRM middleware subsystem

TMOSM was devised to support the execution of TMOs with the use of a minimal set of computing and communication resources while satisfying all the timeliness requirement specifications embedded in TMOs. Once the high-level specifications of timeliness requirements are registered with TMOSM, then the middleware does its best to meet the specifications by using the CPU scheduler and other resource schedulers in the underlying node OS and the network infrastructure. While devising the TMOSM architecture, an emphasis was on making both the analysis of the worst-case time behavior of the middleware and the analysis of the execution time behavior of application TMOs as easy as possible without incurring any significant performance drawback. As a result, use of mechanisms such as semaphore which leads to frequent blockings of threads inside the middleware was avoided completely and instead, a new extension of the Non-Blocking Writer mechanism invented by Kopetz [Kop97], called the Non-Blocking Buffer (NBB) mechanism [Kim05], was used extensively.

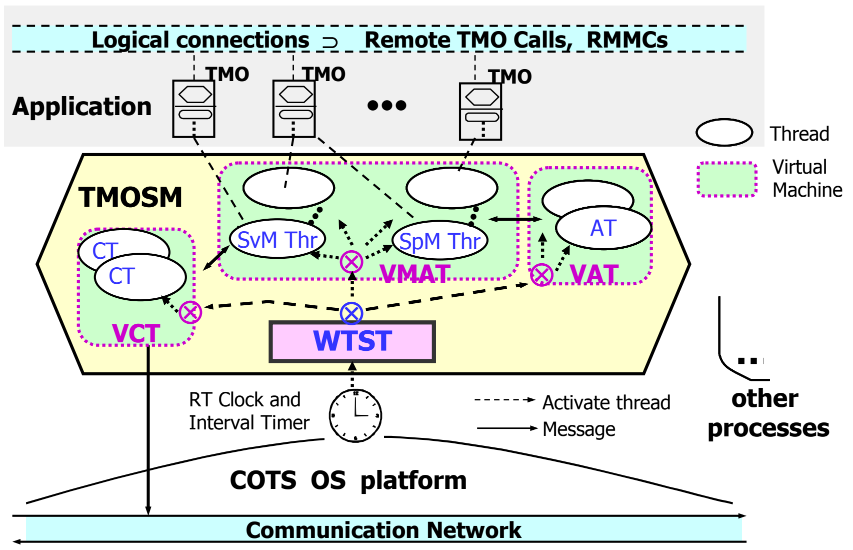

Figure 6. Virtual machines and threads in TMOSM

As depicted in Figure 6, within TMOSM, the innermost core is a super-micro thread called the WTST (Watchdog Timer & Scheduler Thread). It is a "super-thread" in that it runs at the highest possible priority level. It is also a "micro-thread" in that it manages the scheduling / activation of all other threads in TMOSM. Even those threads created by the node OS kernel before TMOSM starts are executed only if WTST allocates some time-slices to them. Therefore, WTST is in control of the processor and memory resources with the cooperation of the node OS kernel.

WTST leases processor and memory resources to three virtual machines (VMs) in a time-sliced and periodic manner. Each VM can be viewed conceptually as being periodically activated to run for a time-slice. Each VM is responsible for a major part of the functions of TMOSM. Each VM maintains a number of application threads. In fact, whenever WTST assigns a time-slice to a VM, the VM in turn passes the time-slice onto one of the application threads belonging to itself. The component in each VM that handles this “time-slice relay” is the application thread scheduler. For example, VM-A has the application-thread-scheduler VM-A-Scheduler. The application thread scheduler is actually executed by WTST. To be more precise, at the beginning of each time-slice, a timer-interrupt results in WTST being awakened. WTST then determines which VM should get this new time-slice. If VM-A is chosen, WTST executes VM-A-Scheduler and as a result, an application thread belonging to VM-A is activated to run for a time-slice as WTST enters into the event-waiting mode.

The set of VMs is fixed at the TMOSM start time. One iteration of the execution of a specified set of VMs is called a TMOSM cycle. For example, one TMOSM cycle may be: VCT VMAT VAT VMAT. The following three VMs handle the core functions:

(VM1) VCT (VM for Communication Threads): The application threads here are those dedicated to handling the sending and receiving of middleware messages.

(VM2) VMAT (VM for Main Application Threads): The application threads here are those dedicated to executing methods of TMOs with maximal exploitation of concurrency. In every one of our prototype implementations of TMOSM, the application thread scheduler in VMAT uses a kind of a deadline-driven policy for choosing an MAT to receive the next time-slice [Kim02b].

(VM3) VAT (VM for Auxiliary Threads): Auxiliary threads await orders to execute certain application program-segments and such orders come from main application threads in execution of TMO methods.

Also, WTST provides the services of checking for deadline violations and if a violation is found, it provides an exception signal to the user. We believe that structuring of VMs as periodic VMs is a fundamentally sound approach which leads to easier analysis of the worst-cast time behavior of the middleware without incurring any significant performance drawback.

Although TMOSM serves as a convenient starting point, establishment of a desirable IRDRM middleware subsystem requires intensive research on the following issues.

(SM1) Top-level deadlines such as method completion deadlines and deadlines for arrivals of results from remote service methods and start-time-windows for TMO methods are provided by TMO programmers. However, systematic top-down resource allocation requires derivation of intermediate deadlines associated with intermediate milestones falling before the method completion points. A heuristic has been derived and used so far but intense research in this optimization area has not been performed yet.

(SM2) VCT instantiations running on DC nodes schedule their communication threads associated with a shared LAN (i.e., Ethernet) such that message-send operations by DC nodes over the LAN occur in a TDMA (time-division multiple access) fashion. Ways to orchestrate the scheduling activities of VMAT with those of VCT need to be studied.

(SM3) Attempts to adapt TMOSM to PC clusters have just begun and it opens several new issues. First, distribution of TMO components, i.e., service methods, spontaneous methods, and data-member-groups, among multiple CPUs in a DC node, needs to be explored. Secondly, the feasibility of distributing TMO components among multiple nodes in a cluster need to be investigated thoroughly. A 32-node cluster built by co-author Jenks over the past several years is one of our experimental research tools here [Jen02].

(SM4) With the current TMOSM architecture, TMOSM instantiations running on different DC nodes cooperate and interact frequently among themselves. Even though the TMOSM architecture has been validated extensively in LAN environments over the past several years, its extension to fit into the RT DVC requires substantial changes in the patterns of interaction among its instantiations. This is due to the large communication latency inherent in an RT DVC occupying a large geographical region. The current TMOSM is expected to show rather poor performance when it is ported to such RT DVC without substantial refinement. We feel that a promising research direction in this area is to add another attribute of fundamental nature to the TMO model.
As building-blocks of RT local-area DC application systems, TMOs were treated as all equal neighbors. This notion of distance-unaware TMOs needs to be changed to derive a newly extended TMO model called the *distance-aware TMO* (*DA-TMO*) which is effective in constructing RT wide-area DC systems. Then DA-TMO programmers should expect that TMOSM instantiations supporting nearby TMOs will interact with a relatively high frequency whereas TMOSM instantiations supporting TMOs separated by long distances will interact less frequently. They should also expect that a call by a client TMO for a service offered by a remote TMO can involve searches for information not readily available in the local TMOSM instantiation. TMOSM can now be aware of when an RMMC covers a large geographical area.

6. Pipelined execution model for RT wide-area DC

A *coarse-grain pipelined execution model* will allow simultaneous use of DC resources without the long waits associated with distributed synchronization. Such an execution model, in which a series of computational stages can be performed concurrently with the results of one stage being passed on to the next stage, is known, but not commonly used in conventional scientific computing [Cul99]. Rather than a strict linear pipeline, a multi-path pipeline may be used if the application warrants it.

For example, consider the following scenario. Data is acquired by an electron microscope and fed into the local cluster computer CC1 located in San Diego, where it may be preprocessed and/or stored for archival purposes. From there, it is sent to two remote clusters, CC2 and CC3, for further processing. At the cluster CC2, perhaps more data from a local sensor could be added to the data stream. The two "processing" clusters CC2 and CC3 perform some (presumably different) processing on the data and then pass it on to the visualization cluster CC4 located in Irvine, which further processes the pre-digested data and presents it to researchers via display panels. Each of these steps takes place concurrently with different chunks of data being sent from San Diego to Irvine. This allows us to visualize data from one period of measurement time, while the data from the next period is being processed and the data from the period after that is being acquired. CC4 may also perform a remote control function related to the electron microscope in San Diego and other sensors in other regions, in which case RT control is taking place over a long distance.

In order to orchestrate this DC pipeline with its high-volume data transfers and at the same time provide RT service guarantees, we need to manage and schedule all the cooperating clusters and communication paths forming the pipeline together for timely computation and communication actions. An extension of the IRDRM subsystem to support such a pipelined execution model is a non-trivial challenge.

To operate the pipelines efficiently, pipeline stages spread over wide areas must act synchronously and this requires use of a good-quality globally referenced time base. The communication between clusters must also show highly predictable timing behavior. In principle, the middleware models and the DA-TMO programming model discussed in preceding sections should enable efficient programming of such wide-area pipelines. When the implicit (message-less) time-based synchronization and other forms of TCoDA are properly exploited, a program on one cluster will know in advance when to expect the next message from the other clusters.

Another approach potentially beneficial in such pipeline programming is to apply within each cluster the Single-Program Multiple-Data (SPMD) programming model [Dar86], in which the same program runs on all nodes, but each node can execute a different path through the program based on the node's identity, program inputs, and other factors. An interesting approach to explore here is to integrate the SPMD programming model with the DA-TMO programming model toward the goal of making the wide-area pipeline programming relatively easy.

7. Conclusion

By composing a real-time wide-area distributed computing middleware framework from demonstrably deterministic components, including OptIPuter-enabled dedicated light paths, TT-Ethernet switches, and TMO middleware, we have employed unique technology to design deterministic infrastructure that was not possible until now. This RT DVC will enhance Grid and DVC mechanisms for allocation of resources, including computers and network links, with timeliness requirements at the forefront. Applications that run on RT DVCs will use timing-based specifications to define their real-time requirements, which makes programming logical and reasonable. Our design accommodates the many millisecond delay inherent in long-distance links and will be able to schedule distant events to rendezvous precisely and will support multiple messages in-flight when latency exceeds transmission period. The framework will provide unprecedented support for wide-area globally coordinated actions based on time and events – capabilities now only available for smaller real-time systems.

Acknowledgment: The research reported here is supported in part by the NSF under Grant Numbers 02-04050 (NGS) and 03-26606 (ITR), and in part by the NSF under Cooperative Agreement ANI-0225642 to UCSD for "The OptIPuter". No part of this paper represents the views and opinions of any of the sponsors mentioned above.

References

[Cul99] D. E. Culler, J. P. Singh, and A. Gupta, '*Parallel Computer Architecture: A Hardware/Software Approach*', San Francisco, Morgan Kaufmann, 1999.

[Dar86] F. Darema-Rogers, D. A. George, V. A. Norton, and G. F. Pfister., "Single-Program-Multiple-Data Computational Model for Epex/Fortran", IBM T. J. Watson Research Center, Yorktown Heights, Technical Report RC 11552, Nov. 1986.

[Fos97] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", *Int'l Jour. of Supercomputer Applications*, vol. 11, no. 2, pp. 115-128, 1997.

[Fos98] I. Foster, and C. Kesselman (Eds.), '*The Grid: Blueprint for a New Computing Infrastructure*', San Francisco, CA, Morgan Kaufmann, 1998.

[Fos01] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *Int'l Journal of Supercomputer Applications*, vol. 15, no. 3, 2001.

[Jen02] S. Jenks, "How to Build a Rackmount Linux Cluster", July 3, 2002, available from http://www.ece.uci.edu/~sjenks/Cluster/index.html.

[Kim97] Kim, K.H., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, August 1997, pp.62-70.

[Kim99] Kim, K.H., Ishida, M., and Liu, J., "An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation", *Proc. ISORC '99 (IEEE CS 2nd Int'l Symp. on Object-oriented Real-time distributed Computing)*, May 1999, pp.54-63.

[Kim00] Kim, K.H., "APIs Enabling High-Level Real-Time Distributed Object Programming", *IEEE Computer*, June 2000, pp.72-80.

[Kim02a] Kim, K.H., "Commanding and Reactive Control of Peripherals in the TMO Programming Scheme", *Proc. ISORC 2002 (5th IEEE CS Int'l Symp. on OO Real-time distributed Computing)*, Crystal City, VA, April 2002, pp.448-456.

[Kim02b] Kim, K.H., and Liu, J.Q., "Going Beyond Deadline-Driven Low-level Scheduling in Distributed Real-Time Computing Systems", in B. Kleinjohann et al. eds., '*Design and Analysis of Distributed Embedded Systems*' (Proc. IFIP 17th World Computer Congress, TC10 Stream, Montreal, Can., Aug 2002), Kluwer, pp.205-215.

[Kim04] Kim, K.H., "Wide-Area Real-Time Distributed Computing in a Tightly Managed Optical Grid- An OptIPuter Vision", *Proc. AINA 2004 (Proc. 18th Int'l Conf. on Advanced Information Networking and Applications)*, Fukuoka, Japan, March 29 - 31, 2004, Volume 1, pp.2-11 (Keynote paper).

[Kim05] Kim, K.H., "A Non-Blocking Buffer Mechanism for Real-Time Event Message Communication", to appear in *Real-Time Systems - The International Journal of Time-Critical Computing Systems*.

[KimH02] Kim, H.J., Park, S.H., Kim, J.G., and Kim, M.H., "TMO-Linux: A Linux-based Real-time Operating System Supporting Execution of TMOs", *Proc. ISORC '02 (IEEE 5th CS Int'l Symp. on Object-Oriented Real-time Distributed Computing)*, Washington D.C., April 2002, pp. 288-294.

[Kop97] Kopetz, H., '*Real-Time Systems: Design Principles for Distributed Embedded Applications*', Kluwer Academic Pub., ISBN: 0-7923-9894-7, Boston, 1997.

[Kop05] Kopetz, H., Ademaj, A., Grillinger, P., and Steinhammer, K., "The Time-Triggered Ethernet (TTE) Design", to appear in Proc. *ISORC 2005 (IEEE CS 8th Int'l Symp. on Object-oriented Real-time distributed Computing)*, Seattle, WA, May 2005.

[Sma03] Smarr, L.L., Chien, A.A., Defanti, T., Leigh, J., and Papadopoulos, P.M., "The OptIPuter", *Comm. ACM*, Nov. 2003, Vol. 46, No. 11, pp.59-67.