To appear in Proc. AINA-2004, March 2004, Fukuoka, Japan,
IEEE CS Press.

Wide-Area Real-Time Distributed Computing in a Tightly Managed Optical Grid - An Optiputer Vision

K. H. (Kane) Kim

DREAM Lab, Dept. of EECS
University of California
Irvine, CA, 92697 U.S.A.
http://dream.eng.uci.edu

(Keynote)

Abstract: Wide area network (WAN) -based distributed computing (DC) has become an active field of research, especially under the label of grid computing. On the other hand, research on WAN-based real-time (RT) DC has remained in an embryonic stage. The situation is changing, however. Continuous increase in the availability of optical channels, called *lambdas*, along with steady decrease of their costs, have given rise to efforts to establish optical network infrastructure in which the possibility of dynamically allocating entire end-to-end light-paths to different message streams created by a moderate number of important applications is real. With such infrastructure there is no major obstacle in facilitating RT DC. The notion of RT distributed virtual computer (DVC), which can also be viewed as an RT sub-grid, is introduced. Then the challenging issues in adapting some promising technologies established for RT DC in LAN environments to enable realization of RT DVCs in WAN environments are discussed. Programming model and resource management are the focus of the discussion.

Keywords: real time, grid, optical, network, wide area, WAN, latency, guaranteed bandwidth, resource management, programming model, global time, object, TMO.

1. Introduction

More than a half century ago the computer system technology started with the uni-processor-based system technology. Then it has been expanded gradually over subsequent years to encompass the multi-processor-based system technology, the local area network (LAN) -based distributed computing (DC) system technology, and finally, the wide area network (WAN) -based DC system technology. In accordance with that evolution, the real-time (RT) computing system technology followed the same path but about 10 years behind the evolution of the main-stream non-RT computing system technology.

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 time-windows. So far, the RT computing system technology has not been expanded far enough to enable a broad range of *WAN-based RT DC*, although cases with limited functionality and quality-of-service (QoS) may have appeared sporadically in recent years.

Recently, WAN-based DC has become an active field of research, especially under the label of *grid computing*. The term "grid computing" has been used broadly to mean "coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations" [Fos01]. Foster adds in [Fos02] that grids are a class of infrastructure built on the Internet and the world wide web (WWW) that

(1) coordinates resources that are not subject to centralized control,
(2) uses standard, open, general-purpose protocols and interfaces, and
(3) delivers non-trivial QoSs.

As the grid computing research community grows, the branch of the community which deals seriously with the QoS aspect is showing increasing interests in the potential for wide-area RT DC. In a sense, the main interests of the RT computing research community are in exploring extremely high levels of QoS. They would often like to ensure that the probability of a node missing a *completion deadline* for a certain important output action is negligible. Therefore, they are seriously concerned with the ease of determining *tight bounds on message communication latency* and *guaranteed progress rates of threads*, not merely average performance indicators such as throughput of a communication channel.

It is then clear that an *RT grid* (or an *RT sub-grid*) is a grid (or a sub-grid) which facilitates message

communications with *easily determinable tight latency bounds* and computing node operations enabling *easy guaranteeing of timely progresses of threads toward computational milestones*. What has prevented the creation of a broad range of RT grids ? The issue of operating computing nodes to carry threads to reach computational milestones in time is not unique to RT grid computing or WAN-based RT DC. It has been extensively studied in the context of LAN-based RT DC.

The issue of transferring messages with easily determinable tight latency bounds has been the main stumbling-block. The issue is not so much of inventing new communication architectures and protocols. It is rather an economic constraint. Due to economic constraints, many communication links in the Internet have been operated to allow *fully flexible unconstrained sharing by an unpredictable dynamically fluctuating mix of message streams*. If we were to achieve message communications with easily determinable tight latency bounds, it is inevitable to put some restrictions on the manner in which communication links or paths are shared. This means reduced overall utilization of communication links and paths, which has been so far considered to be too expensive in the general Internet field.

The situation is changing, however. Continuous increase in the availability of wavelengths of light, called *lambdas*, traversing strands of optical fiber while the costs of acquiring such lambdas are steadily decreasing, have encouraged some technological visionaries to propose to let major science laboratories spread over multiple cities and states acquire dedicated lambdas and use them to form fully optical data paths among them. An *OptIPuter* project which started in 2002 under the sponsorship of US NSF with headquarters at University of California, San Diego (UCSD) and University of Illinois at Chicago (UIC) and four additional universities (Northwestern University, San Diego State University, Information Science Institute at University of Southern California, and University of California, Irvine) participating, is a conspicuous example [Sma03]. According to Smarr et al, "the OptIPuter is an envisioned infrastructure, available within five years (i.e., by the end of 2008), tightly coupling computational resources over parallel optical networks using the IP communication mechanism."

With infrastructure such as OptIPuter, the possibility of dynamically allocating entire end-to-end light-paths to different message streams emerges. The jitter in communication delays becomes then a non-issue. Somewhat more flexible sharing of lambdas without incurring excessive delay jitters can also be explored. All these research efforts are based on the premise that infrastructure with the aforementioned characteristics can grow to accommodate hundreds or thousands of sites in ten or so years.

In WAN environments with moderate bounded communication delay jitters, there is no major obstacle in facilitating RT DC. Therefore, the author and his collaborators are exploring RT DC potentials in a tightly managed optical grid, i.e., the OptIPuter environment. The challenge is how to adapt the most advanced and effective technologies established for RT DC in LAN environments to WAN environments such as the OptIPuter environment. Although a major obstacle does not exist, there are many questions related to achievable DC performance, optimal use of resources, newly enabled applications, etc.

This paper is a summary of what the author considers to be the major research issues in the way of such extrapolation of RT DC technologies. The paper starts with a brief discussion in Section 2 on the new - generation RT applications benefiting from WANs with well controlled latency. Then in Section 3, the notion of an RT sub-grid is introduced and the major component technologies needed to realize such RT sub-grids are discussed in Section 4. Some existing technologies which were established for LAN-based RT DC and appear to be good choices as starting points for development of technologies desired for WAN-based RT DC, are discussed in Section 5. Section 6 provides a conclusion.

2. New-generation RT applications benefiting from networks with well controlled latency

Numerous new-generation RT applications that become enabled with emergence of facilities such as OptIPuter are conceivable. As an example, Figure 1 depicts an application scenario related to earth-quake monitoring and related RT control of sensitive manufacturing plants. Each earthquake monitoring station sends its pre-analyzed sensing reports to one or more earthquake response control centers (ERCCs). Each ERCC collects and 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, still mills, 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 grids, this kind of applications cannot be realized.

Another example is a multi-party multimedia based collaboration. Currently, the quality of a two-site video-conferencing over the Internet facilitated by low-cost facilities (e.g., Microsoft Messenger and other commercial products) seems acceptable to some users but there is no doubt about the desire of all users to see major improvements in quality. In the case of a 3-site or 4-site video-conferencing, let alone a 10-site or 20-site case, the tools with moderate costs and quality acceptable to many

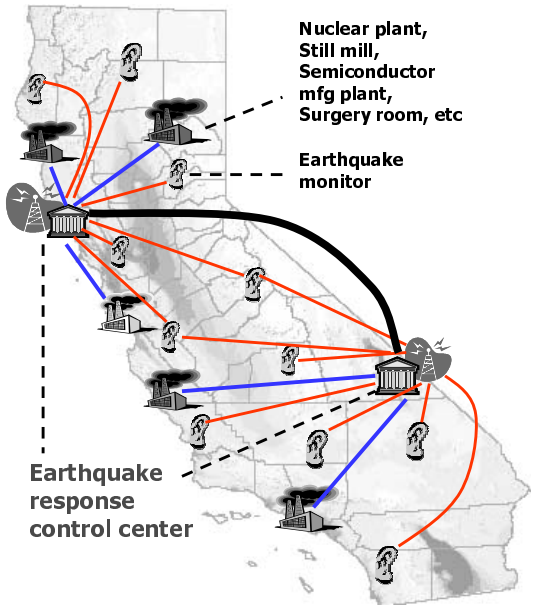

Figure 1. An earthquake monitoring and coordinated response application

users are not simply available. The lack of a highly-reliable RT grid is the main obstacle.

With an RT grid, one may even be able to realize an opera involving multiple play sites in manners making the locations of the sites transparent to the audience sitting in each site. For example, an opera scene visible to the audience in a theater in Irvine may be composed of the scene of the stage on which local performers act plus a screen displaying the scene of the stage in a theater in San Diego. Similarly, the audience in the San Diego theater will be watching the composition of the scene of the local stage and a video-display of the scene of the remote stage in Irvine. The performers in both theaters must act in

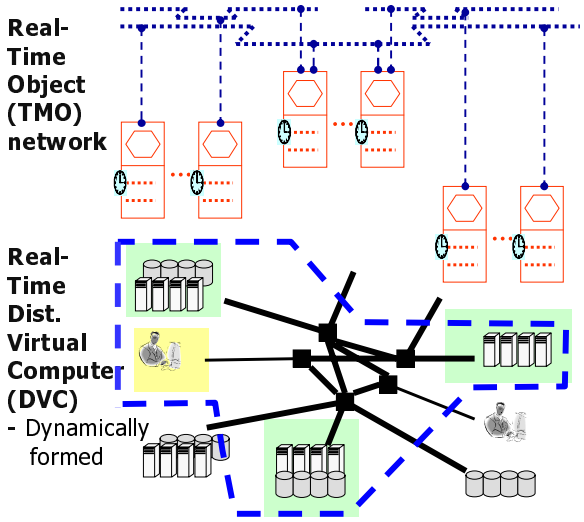

Figure 2. A real-time DVC

synchrony by listening to the music played in both sites synchronously. Careful synchronization of both music plays and video plays in both sites, which is arranged after considering the effect of the message communication delays, is an essential requirement to make this kind of applications successful.

It is easy to find many practical applications in the national defense area. The situations where various regional commands must collaborate in sensor data collection, decision-making, and responsibility sharing and splitting within the time constraints which cannot be met without facilities such as OptIPuter, are numerous.

3. RT Distributed Virtual Computer (DVC)

In the OptIPuter project, Larry Smarr and Andrew Chien at UCSD established the notion of *distributed virtual computer* (DVC) [Sma03]. A DVC is depicted in Figure 2 and has the following characteristics.

(a) A DVC is an image of a distributed computing facility presented to an 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.

(b) 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 notion of DVC, its specialized case for RT computing, i.e., *RT DVC*, can be established. An RT DVC must be distinctively stronger than a general DVC in the following characteristics:

(a) 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 DVCs, it must be operated to provide both tight latency bounds and guaranteed bandwidths to the RT DVC.

(b) 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 relative easy procedures for assuring the very high probability of application processes and threads reaching important milestones always in time. This means that computing nodes must be equipped with appropriate infrastructure software, i.e., OS kernel and middleware with relatively easily analyzable QoS.

(c) 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.

Given an RT DVC, an RT DC application can be established on it by producing the application software in a form which yields a relatively easy interpretation of resource demands imposed by it. Such modern high-reliability *RT distributed programming* is currently an active area of research even for the case of LAN-based RT DC [ISO, WOR].

4. Component technologies needed to realize RT DVCs

In order to realize RT DVCs in cost-effective manners, a few new component technologies applicable to optical grids based on dedicated lambdas need to be established.

4.1 Optical path management layer

Various research efforts are under way to develop efficient ways for establishing and managing lambda-based message paths connecting distant sites. Some deal with new protocols, including both the protocols requiring enhanced routers (e.g., XCP) and those working on existing routers (e.g., RBUDP, GTP, SABUL) [Fal03]. There are also efforts dealing with switching hardware mechanisms, e.g., optical switches [Fal03].

In addition to the aforementioned efforts, those aimed for making it easier to analyze the latency bounds are needed. Searches for additional protocols which may yield better predictable and controllable end-to-end latency may be worthwhile. In the case where an RT DVC acquires all needed end-to-end light-paths in the exclusive-use mode, i.e., it does not share any light-path with any other DVC or external entity, then all that is required is to develop an intra-RT-DVC middleware subsystem responsible for managing level-1 message paths needed within the RT DVC in well disciplined manners. There are currently no standardization efforts which will impose any significant constraint on such developments.

On the other hand, if an RT DVC were to share some light-paths with other DVCs, then such light-path management becomes a subject of broader concerns and more complex trade-offs.

4.2 Middleware for management of resources within an RT DVC

Creation of an RT DVC will involve the services of the OptIPuter middleware. Such services may be realized as extensions or refinements of the services defined in earlier grid computing research, e.g., GRAM service in Globus [Fos01]. Within an RT DVC, multiple applications may dynamically share available resources. Therefore, a middleware subsystem responsible for allocating those resources within the DVC as demands from the applications arise, needs to be developed.

It seems reasonable to attempt to develop such an *intra-RT-DVC resource management* (IRDRM) middleware subsystem as an extension of a well established middleware subsystem effective in LAN-based RT DC systems. In Section 5.2, one such candidate middleware subsystem will be briefly reviewed. The necessary extension will be primarily in tuning the middleware subsystem to perform well in the optical grid environment by reflecting increased communication latency of the optical grid.

Research over the past three decades has indicated that 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. On the other hand, the latter step, i.e., reasonably 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. Research on high-level styles of specifying RT DC programs started only in 1990's and the scope of such research has been limited and the progress has been slow [Bol00, Kim97, ISO, OMG02, ISO].

Promising high-level programming styles which have been studied so far have all taken the form of one kind or another of *RT DC object programming*. Such programming produces RT DC applications structured as networks of RT objects. Specifications of RT objects must include specifications of parameters which can be easily interpreted by the node kernel and an intra-RT-DVC resource management middleware subsystem and lead to efficient resource management with respect to achieving application goals. Structuring and use of such parameters is still an immature subject of active research but is expected to mature during this decade whereas before the notions of high-level RT objects emerged in 1990's, finding such parametric specification schemes could not have been attempted. This subject will be discussed further in the next section and Section 5.2.

Therefore, a desirable IRDRM middleware subsystem must support RT DC application programs structured as networks of RT objects. It should be able to deduct short-term resource demands from the parametric specifications easily and efficiently and make efficient resource allocations.

It is also conceivable that the middleware subsystem may evolve to incorporate capabilities for expanding the pool of resources available within the RT DVC by acquiring new resources from the OptIPuter-wide resource management middleware system as needs arise. However, such an extension is not a small research step.

4.3 High-level programming model and programming interfaces

New-generation RT DC applications running on RT optical grids are bound to be highly complex relative to current software engineering practices. These applications must be designed in forms which are among the best in terms of enabling cost-effective analysis, validation, and verification. At present the most promising branch of RT programming that enables such cost-effective analysis and validation is the high-level RT DC object programming which has been demonstrated to considerable extents in LAN-based RT DC environments [Kim97, Kim00, OMG02].

It was mentioned in the preceding section that another important benefit of using the high-level RT DC object programming approaches is in enabling efficient and systematic use of resources.

The high-level RT DC object programming schemes established needs to be strengthened in at least three areas to support WAN-based RT DC effectively. First of all, they lack special features for making efficient use of computing resources in tightly coupled PC clusters. Secondly, coordinated scheduling of object method executions by CPUs, IO activities, and network communication activities has been insufficiently developed [Kim01b]. Thirdly, how the long communication latency in optical grids will impact the distributed synchronous operations has not been much studied. These issues will be discussed further in the next section.

5. Initial base for realization of an RT DVC programming model and an IRDRM middleware subsystem

As discussed in Sections 4.2 and 4.3, it is very important to adopt a good programming model with respect to making an RT DVC to be easy and safe to program and show good performance. Before discussing our technical base from which to start developing an RT DVC programming model, the evolution of RT programming models over the past 40 years is briefly reviewed first.

5.1 Evolution of RT programming models

5.1.1 RT programming in assembly languages, C, and dialects of PL/1 and Pascal

Before 1990's, RT programming was practiced almost fully by use of assembly languages, C, or dialects of PL/1 and Pascal. Such practices are called here RT programming in low-level styles or C styles.

Initially RT programming dealt with logically simple supervision of peripherals in the polling or the interrupt-driven mode. As the amount of supervisory computations required grew and varied widely among different peripherals, the ineffectiveness of structuring all supervisory computations into a single centralized sequential program became evident. Also, the growing complexity of the data fusion logic, especially, the logic of handling data fusion during nested interrupt handling, provided further impetus for seeking the paths for decentralization of the coordination and supervision of peripherals' activities. From these situations emerged the approach of structuring supervisory computations, including data fusion, into concurrent processes.

Concurrent process structuring of RT programs started with an approach based on an extremely simplistic process structure. Each RT process was a periodic iteration of the device's_request-supervisory_ computation-actuator_command (RSA) sequence. For about three decades, a great majority of research efforts dealt with the following severely restrictive types of processes

(r1) Each process is an *isolated process* and thus there are no information dependency among the processes.

(r2) The *iteration period* is a constant P. A device's request event occurs every P seconds.

(r3) One instance of the 3-step sequence, RSA, is always assumed for every iteration. Nothing like R-S-R-S-A, R-S-A-R-S-A, etc., is considered.

(r4) The maximum time allowed for completion (MTAC) of one R-S-A sequence is usually the same as P. That is, the *completion deadline* is usually the same as the request-arrival-time plus P.

It turns out that even with these severe restrictions, optimal scheduling of processes for running on processors was not easy. Only the cases where only one processor is used and the overhead involved in switching from one process to another is negligible, could be easily analyzed. However, if only one processor is used and the process switching overhead is negligible but one of the four restrictions mentioned above is removed, then the analysis of optimal scheduling algorithms becomes intractable again.

Most engineers then adopted a highly simplistic scheduling scheme, i.e., the *fixed-priority design and priority-driven scheduling* scheme [Fin67, Liu73]. It became known rather early that the *deadline design and deadline-driven scheduling* scheme is more efficient than the fixed-priority scheme if the overhead incurred in process switching is negligible [Ser72, Liu73].

Still, the practitioners did not use the deadline-driven scheme much at all. One reason is because often they were not sure whether the restriction (r4) about the arrival of the next request being the deadline for handling

the preceding request was reasonable or not in the applications which they were faced with. In the case of the fixed-priority scheme, it did not matter as much since all that the designers were caring was to determine the priority order for the processes and it looked reasonable to them to determine the priority order based on the request interval P even if it was not obvious to them what the reasonable deadline values should be. Secondly, OS designers did not provide in their OSs mechanisms through which application processes could pass the deadline information. Thirdly, if the restriction (r4) was violated, then it was not easy to quantitatively analyze the efficiency advantage of the deadline-driven scheduling over the fixed-priority scheduling.

The next step taken by some researchers was to introduce the possibility of data sharing or message exchanges among the processes [Sha90]. However, a substantial number of researchers introduced it into the fixed-priority processes and the fact that fixed-priority processes are overly simplistic unreasonable designs of processes in such cases was not obvious at that time. The case of interacting processes associated with deadline specifications rather than priority specifications was hardly studied until late 1980's.

Structuring cooperating processes as fixed-priority processes was an ineffective approach. One of the reasons is because it opened the possibility of a high-priority process waiting for data or messages from low-priority processes and the amount of such waiting time could become very large when a low-priority process supplying a message is preempted by another process of which the priority is higher than that of the preempted process but lower than the waiting process. Attempts to mechanically mitigate the damaging impacts of such situations resulted in approaches such as the priority inheritance protocol but the effects were still generally poor [Sha90, Kop97].

The real problem originated from the restrictive nature of the fixed-priority process structure. One needs to start with more reasonably structured processes, e.g., processes consisting of segments associated with different priorities, or for an even better one, processes associated with *milestone deadlines* and *utility functions* [Jen85], processes associated with milestone deadlines and *penalty functions* [Kim01b], etc.

In 1990's, researchers saw the problems mentioned above clearly and started looking for fundamentally different approaches which were aimed for high-level programming styles and involved object-oriented programming languages.

5.1.2 RT process programming in C++

As C++ started gaining popularity in 1990's, RT computing researchers also started exploring possible uses of C++ in RT programming. Initial efforts for rewriting RT processes previously written in C into new versions in C++. Some data structures and related access routines were restructured into object classes. Writing new RT processes in C++ was not much different. The approach for structuring of RT processes remained essentially the same but structuring of monopolized or shared data as objects was a new thing.

Practitioners saw some advantages of writing RT processes in C++ over writing in C. The understandability of C++ RT processes was somewhat better than that of C RT processes. At the same time, some of them remained concerned with the loss of the execution efficiency incurred in switching over to C++ RT processes.

5.1.3 RT DC object programming in C++

By the beginning of 1990's, many interesting cases of RT systems became DC systems. Researchers started seeing *DC objects*, not processes, as highly promising basic building-blocks for RT DC application software. Early efforts sprung up in natural widely scattered forms [Bih89, Ish90, Kim93, Tak92]. In mid-1990's, better organized industry efforts aimed for establishing some standard RT DC object programming approaches and language tools started. RT CORBA [OMG02] and RT Java [Bol00] are the most notable industry efforts. Around year 2000, Microsoft's Embedded .Net technology development started. At this time, all these efforts are still at an early stage and expected to evolve at a rattling rate for many years to come.

In 1992, the author started an academic research project aimed for establishing an RT DC object programming model. The project started with the skeleton of a concrete syntactic structure and execution semantics of a high-level RT DC object named the *Time-triggered Message-triggered Object* (TMO) with the goal of relieving 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 [Kim93, Kim97, Kim00, Kim02, Kim03]. The TMO programming and specification scheme has been enhanced in several steps along with supporting tools (kernel, middleware, API, specification, etc) since then.

TMO facilitates a highly abstract programming style without compromising the degree of control over timing precisions of important actions. In terms of raising the level of abstraction for RT programming, the TMO programming scheme is more advanced than any other practical RT DC programming model established so far. The TMO programming scheme and supporting tools have been used in a broad range of basic research projects and also used in an undergraduate course on RT DC programming at UCI for about three years. However, serious efforts for transferring the technology into industry are about to begin only now.

We have adopted the TMO programming scheme as a highly promising candidate for use in RT grid programming after some necessary extensions. Rationale

for this judgment is given in the next section.

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

The key features of the TMO programming scheme are reviewed first.

(1) TMO has been devised to contain only high-level intuitive and yet precise expressions of timing requirements. No specification of timing requirements in (indirect) terms other than *start-windows* and *completion deadlines* for program units (e.g., object methods) and *time-windows for output actions* is required. Therefore, no priorities and no other indirect and inaccurate styles of expressing timing requirements are associated with TMOs and their methods.

(2) 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.

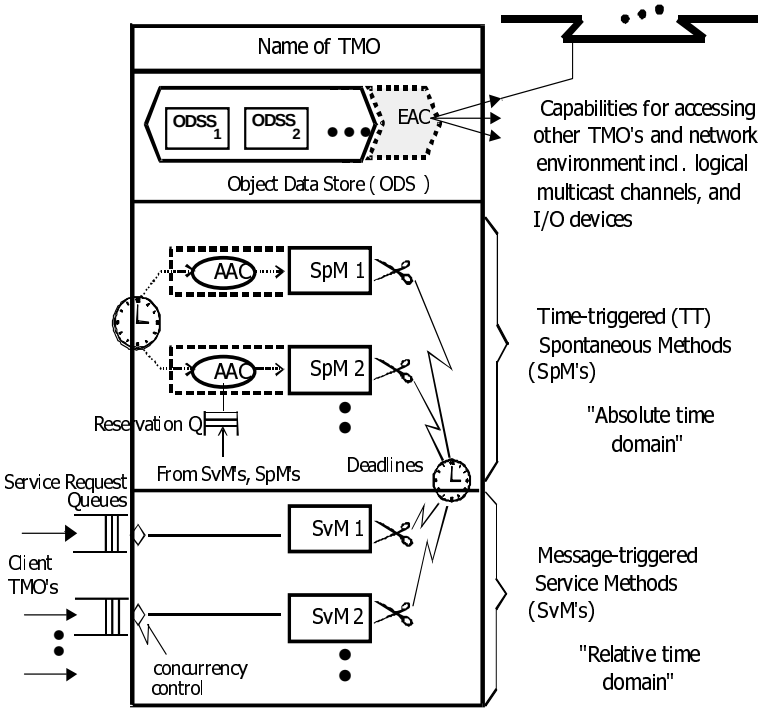

Figure 3. The basic structure of TMO (Adapted from [Kim97])

(3) 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. A TMO instantiation instruction may contain a parameter which explicitly indicates the required precision of the global time base to be established by the TMO execution engine. If a required precision is 1 millisecond, then the statement "do S at 10am" appearing in both TMO1 running on node 1 and TMO2 running on node 2 can create a situation where action S in node 1 occurs truly 1 millisecond before or after action S in node 2 but not a situation where the true time gap between the two occurrences of action S is more than 1 millisecond. If an explicit specification of the parameter is missing, a default value will be assumed by the execution engine. The execution engine raises an error flag if the precision of the global time required by the TMO is beyond the precision that it can provide.

(4) TMO is a natural, syntactically minor, and semantically powerful extension of the conventional object(s). The basic structure is depicted in Figure 3. TMO is a DC component and thus TMOs distributed over multiple nodes may interact via *remote method calls*. Deadlines can be specified in the client's calls for service methods for the return of the service results.

(5) 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) [Kop97] can be easily designed and realized.

(6) TMOs can use another interaction mode in which messages can be exchanged over logical message channels of which access gates are explicitly specified as data members of involved objects. The channel facility adopted in the TMO scheme is called the *Real-time Multicast and Memory-replication Channel* (RMMC) [Kim00], of which an earlier version was called the HU data field channel [Kim95]. The RMMC scheme facilitates RT publish-subscribe channel in one of the most powerful forms. It supports not only conventional event messages but also state messages based on distributed replicated memory semantics [Kop97].

(7) The TMO incorporates deadlines in the most general form. Basically, for output actions and method completions of a TMO, the designer guarantees and advertises execution time-windows bounded by start times and completion times. As mentioned earlier, deadlines can be specified in the client's calls for service methods for the return of the service results.

(8) 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-

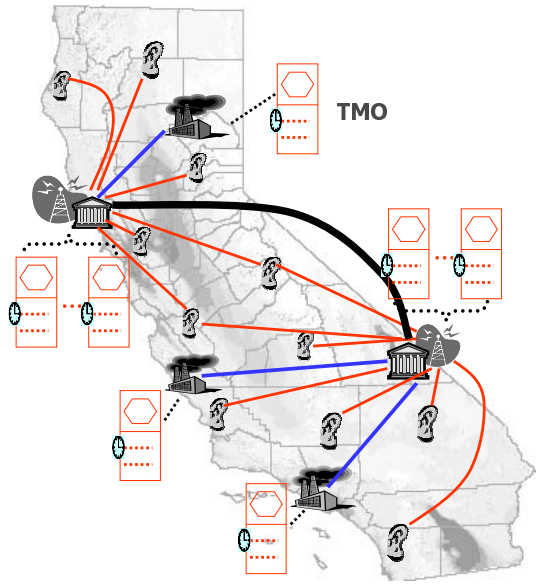

Figure 4. A network of TMOs handling an earthquake monitoring and coordinated response application

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 4.

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

(1) The high-level nature of the programming scheme makes application programming with the RT DVC easier than the programming in other lower-level styles and reading and understanding of the TMO-network-structured DVC application programs easier.

(2) The high-level nature and 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 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.

(3) 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 asynchronous hand-shaking message based coordination, this TCoDA approach can be an order-of-magnitude more efficient and reliable. For example, the two end-points can be designed to simultaneously start at 10AM to take a certain course of actions without exchanging any synchronization messages if they observe certain conditions by 9:59AM.

However, so far experimental research on the TMO programming scheme has hardly included the use of PC clusters, let alone a wide area network of PC clusters. We believe that a proper direction for exploiting potential computing power of the OptIPuter is to apply the *macro-pipeline parallel processing model* across a WAN of cluster computers and the SPMD (Single Program, Multiple Data) model [Dar86] within each of the selected cluster computers. The non-negligible message transmission delay between cluster nodes of the OptIPuter is the main cause for relying on the macro-pipeline processing model. In a sense, each lambda is a substantial part of an end-to-end pipeline and serves as a non-negligible chain of pipeline stages. Therefore, future research must include exploration of further possible extensions of the TMO scheme and analysis of whether any of them makes the lives of the RT DVC programmers further improved.

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

We have been enabling TMO programming without creating any new language or compiler. Instead, a middleware architecture called TMOSM (*TMO Support Middleware*) that provides execution support mechanisms and can be easily adapted to a variety of commercial kernel+hardware platforms compliant with industry standards, was established [Kim99, Kim01a]. 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 TMO programmer. The TMOSM architecture was devised to contribute to simplifying the analysis of the execution time behavior of application TMOs running on TMOSM.

A prototype implementation on Windows XP/2000/NT, TMOSM/XP, was developed [Kim99, Kim02]. Our experiences indicate that even this middleware extension of a general-purpose OS (Windows XP) can support application actions with the 10ms-level timing accuracy. TMOES/AnyORB/NT [Kim01a] is another prototype implementation realized in the form of a CORBA service that runs on platforms equipped with Windows NT and an ORB (object request broker) and supports CORBA-compliant application TMOs. A Windows CE based prototype of TMOSM is partially running and under continuous development. In addition, a research group at Konkuk University, Korea, has established an implementation of similar middleware based on a Linux platform [KimH02].

A friendly programming interface wrapping the execution support services of TMOSM has also been developed and named the *TMO Support Library* (TMOSL) [Kim99, Kim00]. It consists of a number of C++ classes. TMOSL empowers C++ programmers with

powerful and natural mechanisms for specification of unique and essential features of TMO programs. Both TMOSM and TMOSL have been used in an undergraduate course on RT DC programming at UCI.

Although TMOSM serves as a convenient starting point, establishment of a desirable IRDRM middleware subsystem requires extensive research to resolve many issues. Some of them are the following.

(1) Attempts to adapt TMOSM to PC clusters have not yet been made and it opens several new issues. For example, efficient mapping of multiple logical channels (i.e., RMMCs) to multi-path networks in tightly coupled PC clusters, is one such issue.

(2) The large communication latency inherent in an RT DVC occupying a large geographical region is expected to have significant impacts on the performance of TMOSM if TMOSM which has evolved in LAN environments is ported to the RT DVC without substantial refinement.

(3) Security enforcement mechanisms have not been incorporated into TMOSM yet. In the case of an RT DVC, security is an important issue. It seems worthwhile exploring the possibility of importing some mechanisms and protocols established by general Grid computing research community [Fos01].

6. Conclusions

New life seems to be forming in the uncultivated research land of WAN-based RT distributed computing (DC). Stimulations come from the technological visionaries who have started efforts to create optical network infrastructure in which the possibility of dynamically allocating entire end-to-end light-paths to different message streams created by a moderate number of important applications is real. Issues in establishing desirable programming models and resource management middleware have been discussed in this paper. Some technologies which have been effective for realizing RT DC in LAN environments and appear to be good choices as starting points for development of desirable technologies for WAN-based RT DC have been reviewed. Many interesting research issues appear to be waiting for attackers.

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

References

[Bih89] Bihari, T., Gopinath, P., and Schwan, K., “Object-Oriented Design of Real-Time Software”, *Proc. IEEE CS 10th Real-Time Systems Symp.*, 1989, pp.194-201.

[Bol00] Bollella, Greg, and Gosling, James, "The Real-Time Specification for Java", IEEE *Computer*, June, 2000, pp. 47-54.

[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.

[Fal03] Falk, A., Faber, T., Bannister, J., Chien, A.A., Grossman, R., and Leigh J., "Transport Protocols for High Performance", *Comm. ACM*, Nov. 2003, Vol. 46, No. 11, pp.43-49.

[Fin67] Mark S. Fineberg and Omri Serlin, "Multiprogramming for hybrid computation", *Proc. AFIPS Conf.*, vol. 31, Anaheim, Nov. 1967, pp. 1–13.

[Fos01] Foster, I., Kesselman, C., and Tuecke, S., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *Int'l J. Supercomputer Applications*, 15(3), 2001.

[Fos02] Foster, I., "What is the Grid? A Three Point Checklist", *GRIDToday*, July 20, 2002.

[Ish90] Ishikawa, Y., Tokuda, H., and Mercer, C. W., “Object-Oriented Real-Time Language Design: Constructs for Timing Constraints”, *Proc. ECOOP/OOPSLA ‘90*, October 1990, pp. 289-298.

[ISO] Series of *Proc. ISORC (IEEE CS Int'l Symp. on Object-oriented Real-time distributed Computing*, IEEE CS Press*: ’98, ’99, 2000, 2001, 2002, and 2003*.

[Jen85] Jensen, E.D., Locke, C.D., and Tokuda, H., "A Time-Value Driven Scheduling Model for Real-Time Operating Systems", *Proc. IEEE CS Symposium on Real-Time Systems*, Nov. 1985.

[Kim93] Kim, K.H. and Bacellar, L.F., “A Real-Time Object Model: A Step toward an Integrated Methodology for Engineering Complex Dependable Systems”, *Proc. 1993 Complex System Engineering Synthesis and Assessment Technology Workshop*, US Navy NSWC, July 1993, pp.56-64.

[Kim95] Kim, K.H., Mori, K., and Nakanishi, H., "Realization of Autonomous Decentralized Computing with the RTO.k Object Structuring Scheme and the HU-DF Inter-Process-Group Communication Scheme", *Proc. ISADS '95 (IEEE CS 2nd Int'l Symp. on Autonomous Decentralized Systems)*, Phoenix, AZ, April. 1995, pp.305-312.

[Kim97] Kim, K.H., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, Vol. 30, No.8, pp. 62-70, August 1997.

[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 for Real-Time Distributed Object Programming", IEEE *Computer*, pp.72-80, June 2000.

[Kim01a] Kim, K.H., Liu, J.Q., Miyazaki, H., and Shokri, E.H., "TMOES: A CORBA Service Middleware Enabling High-Level Real-Time Object Programming", *Proc. ISADS 2001 (IEEE CS 5th Int'l Symp. on Autonomous Decentralized Systems)*, Dallas, TX, March 2001, p. 327-335.

[Kim02] Kim, K.H., "Commanding and Reactive Control of Peripherals in the TMO Programming Scheme", *Proc. ISORC '02* (*5th IEEE CS Int'l Symp. on Object-Oriented Real-time Distributed Computing*), Crystal City, VA, April 2002, pp.448-456.

[Kim03] Kim, K.H., "Basic Program Structures for Avoiding Priority Inversions", *Proc. ISORC 2003 (IEEE CS 6th Int'l Symp. on Object-oriented Real-time distributed Computing)*, Hakodate, Japan, May 2003, pp. 26-34.

[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 2002 (5th IEEE CS Int'l Symp. on OO Real-time distributed Computing)*, Crystal City, VA, Apr. 2002.

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

[Liu73] C.L. Liu and J.W. Layland. "Scheduling algorithms for multiprogramming in a hard real-time environment", *J. ACM*, 20(1):46-61, Jan. 1973, pp.46-61.

[OMG02] Object Management Group, "Chapter 24. Real-time CORBA", in *CORBA Specification, Version 2.6.1*, http://www.omg.org/technology/documents/formal/corba_2.htm, May, 2002.

[Ser72] Serlin, O., "Scheduling of time critical process", *Proc. AFIPS conf.*, Vol. 40, Atlantic City, NJ, May 1972, pp.925-932.

[Sha90] Sha, L., Rajkumar, R., and Lehoczky, J., "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", *IEEE Trans. on Computers*, Vol.39, No. 9, Sept. 1990, pp. 1175-1185

[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.

[Tak92] Takashio, K., and Tokoro, M., "DROL: An Object-Oriented Programming Language for Distributed Real-Time Systems", *Proc. OOPSLA*, 1992, pp. 276-294.

[WOR] Series of *Proc. WORDS (IEEE CS Workshop on Object-oriented Real-time Dependable Systems)*, IEEE CS Press: *'94, '96, '97, '99, '99F, 2001, 2002, 2003, and 2003F*.