

Real-Time Operating Systems

A Hierarchical Resource Management Scheme Enabled by the TMO Programming Scheme

By Kane Kim, Yuqing Li, K.W. Rim, & Eltefaat Shokri

Mar-08

UCI
DREAM Lab



To improve QoSs of RT DC systems

- **High-level approaches for specification and design of RT DC actions needed.**
 - The low-level specification approaches, i.e., those involving explicit manipulation of threads, thread-priorities, and UDP sockets, blur the vision of the software engineer → He / she cannot see easily the overall picture of the sizable RT DC applications and grasp and analyze the timing requirements that should be imposed on various DC actions.
- **Systematic composition of rigorously built and / or sufficiently validated building-blocks** is essential in order to meet the construction economy and product reliability demands from the societies.
 - Impractical if the raw materials are of low-level entities such as threads, thread-priorities, and UDP sockets.
 - Action timings must be specified not only in intuitively appealing easily understandable forms but also in forms enabling systematic composition of proper timing behavior of higher-level assemblies.

Mar-08

UCI
DREAM Lab



Advancing the Technical Foundation for Resource Allocation

- The biggest obstacle:
Lack of high-level DC programming models
 - From old simple DC models which typically consist of largely, if not completely, independent threads that are often attached fixed priority numbers,
very little knowledge useful for effective resource allocation can be extracted.
- Good high-level DC models are bound to provide **information-rich intuitively appealing timing specifications.**

Mar-08

UCI
DREAM Lab



Advancing the Technical Foundation for Resource Allocation

- Resources to be allocated include **both local computation resources and communication resources.**
 - Sizable RT DC systems are structured in multiple layers
 - DC hardware layer
 - OS kernel layer
 - Middleware layer
 - Application layer

Each layer may internally consist of further divided sub-layers.
- => **Hierarchical resource allocation** is a reasonable approach

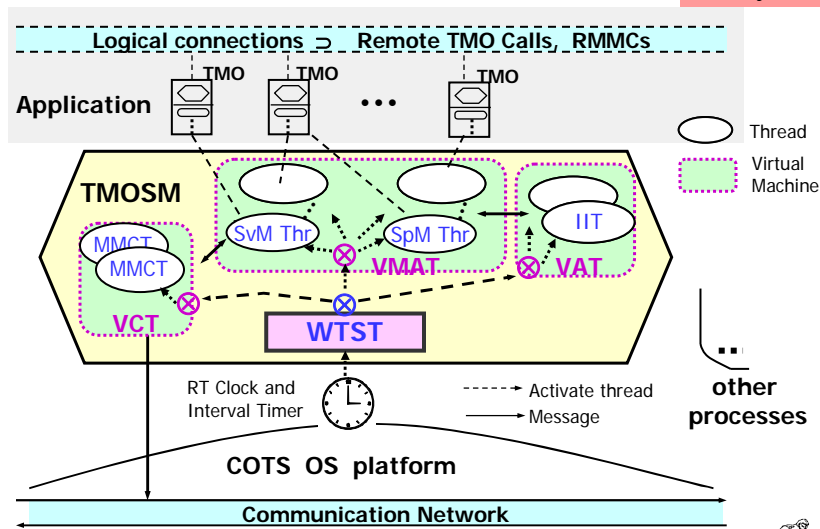
Mar-08

UCI
DREAM Lab



TMO Support Middleware on Windows XP & CE -- TMOSM / XP or CE / Socket

Currently running



Mar-08

UCI
DREAM Lab

Hierarchical Resource Management in TMO-Structured RT DC Environments

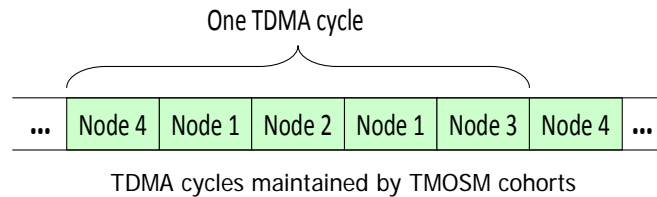
- At the networked system level**
 The *TMO Network Configuration Manager* (TNCM) instantiation in each node cooperates with its peers, i.e., TNCM instantiations in other nodes, for **distributing TMOs to proper nodes** and resolving the competition among the nodes for **shared execution resources such as network bandwidth**.
- At the node level**
 WTST allocates proper **processor cycles** (to be precise, **time-slices**) to each **virtual machine** such as VMAT, VCT or VAT. VCT is also instructed on **how to use the acquired network bandwidth** to support VMAT and VAT.
- At the virtual machine (VM) level**
 The intra-VM scheduler allocates resources to each **ready thread** in its domain.

Mar-08

UCI
DREAM Lab

At the Networked System level: Assignment of application workloads and network access windows to DC nodes

- A proven way to achieve bounded communication delays in an isolated Ethernet environment is to make DC nodes to access the network in a TDMA fashion



- Network bandwidth sharing among the nodes can be adjusted by the TNCM based on run-time information
 - When a computing node crashes
 - No future msg transmission from a certain node expected

Mar-08

UCI
DREAM Lab



At the Node level: Assignment of processor time-slices to VMs and network access windows to different VCT services

- TMOSM adopts a two-level scheduling scheme
 - (L1) VM scheduling
 - Initial schedule of VMs in a node is defined in "config.ini"
 - [VMAT VAT VMAT VCT] => 2/4 of time-slices are given to VMAT
 - (L2) Application thread scheduling
 - WTST calls the Application thread scheduler belonging to the VM to select the next application thread to be executed

Mar-08

UCI
DREAM Lab



At the Node level: Assignment of processor time-slices to VMs and network access windows to different VCT services

- VCT is the sole VM in TMOSM which directly accesses the network.
 - (m1) Messages exchanged among the VMAT peers in DC nodes
 - (m2) Messages exchanged among the VAT peers
 - (m3) Messages originated from a VCT cohort and ending at a VCT cohort.
 - The "config.ini" file may specify the transmission bandwidth usage (TBU) for each class of messages.
 - In a sense, it indicates how many percents of the bandwidth owned by VCT are reserved for supporting the target VM.
 - Also viewed as subdividing a TDMA slot further into sub-slots.
- E.G., VCT's *transmission service cycle* in node A is VUSER1 VUSER2 VMAT VAT VCT, and the TBU specification is 0 10 50 0 40.
- => 50% of the TDMA slots owned should be used for serving VMAT

Mar-08

UCI
DREAM Lab



At the VM level

- Primarily involves the scheduling of application threads (supporting the admitted TMOs) based on the deadlines or laxities associated with the program-segments being carried by the threads.
- Supporting the resource manager at the networked system level on deciding whether to admit a TMO or not.
 - An intra-VM resource manager can check if the admission is safe from the viewpoint of the VM.
When both VMAT and VCT recommend admission of a TMO, the availability of the processor cycles and network access windows needed to support the TMO is guaranteed.
 - A future research topic
 - Identification of various possible safe distributions of TMOs at design time through extensive testing aided by some analyses is the current practical recourse

Mar-08

UCI
DREAM Lab



A Direction for Research on Dynamic Admission of TMOs

- The **processor cycle requirements** that a TMO imposes on VMAT can be estimated on the basis of the timing specifications embedded in TMOs plus the **tight isolated execution time bound (TIETB)** of each method in the TMO.

Here a **TIETB** of a TMO-method is an execution time bound for the cases where the subject method runs alone in the host node without any other TMO or TMO-method co-resident in the same host node in an active mode.

- The timing specifications in the current TMO model include
 - (1) **Autonomous activation conditions (AACs)** associated with SpMs, which include the **guaranteed completion time (GCT)**, also called the **guaranteed execution time bound**,
 - (2) **GCTs of SvMs**, and
 - (3) **Maximum invocation rate (MIR)** associated with each SvM.

Currently, TMO designers are required to specify a GCT for each SpM and one for each SvM but not required to provide a TIETB for each TMO-method.

Mar-08

UCI
DREAM Lab



A Direction for Research on Dynamic Admission of TMOs

- TIETBs are expected to be provided by a tool(s) which analyzes the program code as well as performing test-runs.
- A GCT of a TMO-method > TIETB of the method.
- In setting GCTs in a TMO, the TMO designer reflects the possibility of the subject TMO being co-resident with some other TMOs in the same node.
- MIR is the maximum rate at which the server TMO containing the SvM can receive service requests from client TMOs, e.g., 10 invocations per 100 milliseconds.

Mar-08

UCI
DREAM Lab



A Direction for Research on Dynamic Admission of TMOs

- A simple approach for estimating a tight upper bound on the processor cycle requirements is to compute the bound on the CPU utilization of the target TMO by

$$\sum_{k=0}^{\#ofSpM} \frac{TIETB(SpM_k)}{Period(SpM_k)} + \sum_{k=0}^{\#ofSvM} [TIETB(SvM_k) * MIR(SvM_k)]$$

Here MIR is a normalized value expressed in terms of the number of invocations allowed per unit time.

- If a TIETB of each TMO-method is registered to TMOSM, then TMOSM can potentially tell at the time of the TMO registration (which follows the registrations of all the member methods) whether the TMO can be safely admitted or not.

=> ☒ some inevitable rooms for errors (e.g., false alarms) here → the TMO designer should be provided a means of enforcing admission.

Mar-08

UCI
DREAM Lab



At the VM level

- TMO provides action timing requirements with which a variety of deadline-driven thread scheduling policies can be easily implemented.
- Simple earliest-deadline-first (EDF) scheduling based on GCTs of TMO-methods leads to one undesirable consequence:

When TMO-methods that can all go in parallel produce **intermediate outputs**, i.e., outputs produced in the middle of their executions in contrast to those produced at the end, intermediate outputs of some TMO-methods are greatly delayed while intermediate outputs of some others are produced much earlier.

Major examples of intermediate outputs include
 (1) sending service requests to another TMO,
 (2) generating commands to the actuator devices, and
 (3) updating shared variables (early releases of ODSSs)

Mar-08

UCI
DREAM Lab



At the VM level

- A practical approach to avoid such phenomenon, i.e., great delays in intermediate outputs of some TMO-methods while intermediate outputs of some others are produced much earlier, is to **attach completion deadlines to the intermediate output actions** and let TMOSM reflect those deadlines in scheduling application threads.
To be practical such deadlines should be **mechanically derived from the GCTs for TMO-methods** supplied by the TMO designer.

Mar-08

UCI
DREAM Lab



At the VM level: A simple approach for determining a deadline to be attached to an intermediate output

- Assume that a TIETB of the method-segment that starts from the beginning of the method and ends at the finishing point of the intermediate output is **D1** and a TIETB of the method-segment starting after the intermediate output and covering the rest of the method is **D2**.

The deadline for the intermediate output can be set to:

$$\text{GCT} \times \text{D1} / (\text{D1} + \text{D2})$$

- * Such an approach for setting intermediate deadlines was discussed previously, e.g., [Zha05], but the TMO scheme makes its practical realization truly feasible.
- Other variations are worth exploring in future research.

Mar-08

UCI
DREAM Lab

