Compositional Analysis for the Multi-Resource Server *

Rafia Inam*, Moris Behnam†, Thomas Nolte†, Mikael Sjödin†
*Management and Operations of Complex Systems (MOCS), Ericsson AB, Sweden
†Mälardalen University, Västerås, Sweden
rafiia.inam@ericsson.com
†moris.behnam, thomas.nolte, mikael.sjodin} @mdh.se

Abstract—The Multi-Resource Server (MRS) technique has been proposed to enable predictable execution of memory intensive real-time applications on COTS multi-core platforms. It uses resource reservation approaches in the context of CPU-bandwidth and memory-bus bandwidth reservations to bound the interference between the applications running on the same core as well as between the applications running on different cores. In this paper we present a complete composable local and global schedulability analysis for the Multi-Resource Server technique. Based on the proposed analysis, we further provide an experimental study that investigates the behaviour of the MRS and identifies the factors that contribute mostly on the overall system performance.

Keywords—Hierarchical scheduling, compositional analysis, CPU-bandwidth partitioning, memory-bandwidth partitioning.

I. INTRODUCTION

With the advent of highly efficient multicore architectures, multiple real-time applications are integrated together and are executed concurrently on multicore platforms. As a result, these applications share not only the CPU with each other, but also other physical resources of the multicore architecture (like shared caches, memory-bus bandwidth, and memory). Contention for the shared physical resources is a natural consequence of sharing [1], [2]. It does not only reduce throughput but also affects the predictability of real-time applications.

On unicore platforms, the server-based scheduling has been developed to achieve predictable integration by successfully bounding the interference between integrated applications [3], [4], [5], [6]. However, this approach is CPU centric and is limited in managing the CPU-resource only. It does not account for activities that are located on different cores and thus still allow interference amongst applications in an unpredictable manner. For multicore platforms, a solution has been proposed to solve these problems through updating the traditional server-based scheduling approach with a novel memory aware Multi-Resource Server (MRS) technology [7], [8] for Commercial Off-The-Shelf (COTS) multicore hardware.

MRS enables predictable execution of real-time applications on multicore platforms through resource reservation approaches in the context of CPU-bandwidth reservation and memory-bus bandwidth reservation. The MRS provides temporal isolation, both between tasks running on the same core (through CPU partitioning), as well as between tasks running on different cores (through memory-bus bandwidth partitioning). The latter could, without MRS, cause interference due to contention on the shared memory bus. A local analysis for tasks executing in an MRS considering a constant memory access time has been presented in [7].

Contributions: In this paper we update the local analysis by relaxing this assumption and considering the worst case delay for accessing memory requests in our analysis. We present a complete and composable global schedulability analysis for both resources (CPU- and memory-bandwidth) of the MRS. Further, we provide a study that brings insight on how these both resources relate to each other. In addition, the evaluation shows the effect of changing the priority of a memory-intensive task on both of these resources.

The preliminary work [7] focused on just the presentation of the general idea of the MRS and described its initial local schedulability analysis. It did not address the global schedulability analysis and lacked an investigation study. In this paper we complete the local and the global analysis and perform an experimental study to investigate the behavior of the MRS and the factors effecting its behavior.

Paper Outline: Section II presents the related work on server-based and memory access techniques for multicore systems. Section III explains our system model. The local and global schedulability analysis is described in Section IV. The correlation between (1) CPU and memory budgets, (2) private and shared memory banks, and the impacts of (1) the period of memory-intensive task and (2) the period of the MRS on server-budgets is investigated in Section V, and finally Section VI concludes the paper.

II. RELATED WORK

The problem of contention of shared resources has gained a significant importance in the context of multicore embedded systems. Hierarchical scheduling is one technique to provide predictable execution on unicore platforms [5], [9], [10]. Solutions for multi-core architectures are based on strong (often unrealistic) assumptions on no interference originating
from the shared hardware [11]. For multicore architectures, the assumption no longer remains valid.

Some highly predictable Time Division Multiple Access (TDMA) based techniques are used for memory bus arbitration. Rosen et al. [12] measured the effects of cache misses on the shared bus traffic where the memory accesses are confined at the beginning and at the end of the tasks. Schranzhofer et al. [13] relaxed this assumption of fixed positions. They divide tasks into sets of superblocks that are specified with a maximum execution time and a maximum number of memory accesses. Another work that guaranteed a minimum bandwidth and a maximum latency for each memory access was proposed by Akesson et al. [14] using a two-step approach to share a predictable SDRAM controller. These techniques eliminate the interference of other tasks through isolation; however, they are limited in the usage of only a specified (non-COTS) hardware.

Schliecker et al. [15] have bounded the shared resource load by computing the maximum number of consecutive cache misses generated during a specified time interval. The joint bound is presented for a set of tasks executing on the same core covering the effects of both intrinsic and pre-emption related cache misses. A tighter upper bound on the number of requests is presented by Dasari et al. [16] by using non-preemptive task scheduling. However, these works lack the consideration of independently developed subsystems and the use of memory servers to limit the access to memory bandwidth. The main focus of our work is on both aspects (compositionality and independent development) w.r.t. server-based methods. Pellizzi et al. [17] proposed the division of tasks into superblock sets by managing most of the memory request either at the start or at the end of the execution blocks. This idea of superblocks was later used in TDMA arbitration [13]. All these techniques assume a constant access time for each memory request and do not consider the reordering of requests. Bak et al. presented a memory aware scheduling for multi-core systems in [18]. They used PRedictable Execution Model (PREM) [19] compliant task sets for their simulation-based evaluations. However, they do not consider a server-based method. Further the usage of PREM requires modifications in the existing code (tasks), hence this approach is not compliant with our goal to execute legacy systems on the multi-core platform.

Recent works [20], [21] considered variable access time of memory requests for tasks executing concurrently on different cores and contending for memory accesses. Wu et al. [20] considered private banks for each requestor, using a FIFO ordering for serving requests, by considering one queue for each bank and a global queue to accumulate requests from each bank. The work in [21] provided analysis for both private and shared DRAM banks and considered the First Ready First Come First Serve (FR-FCFS) scheduling policy to account the reordering effect. However, these works considered the task-level schedulability only and lack the consideration of independently developed subsystems and the use of memory servers to limit the access to memory bandwidth, which is our main focus.

A server-based approach to bound the memory load of low priority non-critical tasks executing on non-critical cores was presented in [22]. Memory servers are used to limit memory requests generated by tasks of non-critical cores. A response time analysis is proposed for tasks that are located on critical cores, including the interference that can be generated from non-critical cores, considering a constant access time for memory requests. We propose a more general approach to support both composability and independent development of subsystems by using servers on all cores. The analysis in [22] only considers one memory server on each non-critical core while we present analysis for both time and memory aspects of the servers executing on all cores and consider multiple servers per core. An analysis for variable DRAM access time to serve memory requests is presented in [23], and it is used in the schedulability analysis of our MRS in Section IV.

III. System Model

Here we present our hardware platform, the system model and the assumptions we follow.

A. Architecture

We assume a single-chip multicore processor with a set of identical cores. Each core has a set of local resources; primarily a set of caches for instructions and/or data and busses to access these from the cores. The system has a set of resources that are shared amongst all cores: this is typically a Last-Level Cache (LLC), a main-memory (DRAM) and a shared memory bus. The architectures like Intel core 2 CPU 6700, Intel i5 3550, etc. comply to these assumptions.

In this work we assume that a local cache miss is stalling, which means whenever there is a miss in a LLC, the core is stalling until the cache-line is fetched from memory. We assume that all memory requests from the LLC to the shared DRAM go through the same bus, and that the bus serves one request at a time.

**DRAM:** We assume that the multicore processor uses Double Data Rate Dynamic RAM (DDR DRAM) as their main memory resource [24], which is shared amongst all of the cores. The controller employs First-Ready First Come First Served (FR-FCFS) scheduling policy [25], that prioritizes the ready DRAM commands (row-hit memory requests) over others and for ties, it prioritizes older requests in order to improve row-hit ratio and maximize the overall throughput.

DRAM bank partitioning (or private banks) is considered to divide the banks into partitions where memory request can access one bank in DRAM. Many COTS architectures do not support private banks, but it can be achieved through operating system bank partitioning [26]. We assume both private banks and interleaved or shared banks (where memory request can access all banks in DRAM) are available and only one type can be used at a time similar to [21]. The DRAM model is used to compute the worst case delay for a DRAM memory request. More details on DRAM background, DRAM model, and memory interference delay analysis can be found in [23].

**Worst case delay for a DRAM memory request:** \( D_{\ell} \) presents worst case delay to fetch the data for a single memory
request from DRAM into cache. The $D\ell$ analysis depends on
the hardware architecture and on the number of cores in the
system, private or shared banks, along with the scheduling
policy of memory controller used to serve parallel requests.
It is independent of the maximum number of requests that
can be generated in all the other servers. $D\ell$ is a sum of
(1) worst case service time for a single memory request and
(2) worst case delay this request can be delayed by other
simultaneous requests (generated by other tasks executing on
other cores) served by the memory controller. $D\ell$ is computed
for worst cases for both private (denoted as $D^{\bigtriangledown}$) and shared
DRAM banks (denoted as $D^{\ell}$). A brief overview about
the computations of memory interference delay analysis is
presented in Section IV-A2 and its complete details can be
found in [23].

B. Server model

Our scheduling model for the multicore platform can be
viewed as a set of trees, with one parent node and many
leaf nodes per core, as illustrated in Figure 1. The parent
node is a node scheduler and leaf nodes are the subsystems
(servers). Each subsystem contains its own internal set of tasks
that are scheduled by a local scheduler. The node scheduler
is responsible for dispatching the servers according to their
bandwidth reservations (both CPU- and memory-bandwidth).
The local scheduler then schedules its task set according to
a server-internal scheduling policy.

![Fig. 1: A multi-resource server model](image)

We follow the same model for the MRS. Each MRS $S_s$
is allocated a period and two different budgets, according to
$(P_s, Q_s, M_s)$, where $P_s$ is the period of the server, a CPU
budget $Q_s$ is the amount of CPU-time allocated each period,
and a memory-bandwidth budget $M_s$ is the number of memory
requests in each period. The CPU-bandwidth of a server is
thus $Q_s/P_s$ and we assume that the total CPU-bandwidth is
not more than 100%.

During run-time, each MRS is associated with two dynamic
attributes $q_s$ and $m_s$ which represent the amount of available
CPU- and memory-budgets respectively. For both levels of
schedulers, including the node and server-level, the Fixed
Priority Pre-emptive Scheduling (FPPS) policy is implemented.

We assume that each server is assigned to one core and that
its associated tasks will always execute only on that core, i.e.,
task or server migration is not allowed.

The MRS is of periodic type, i.e., it replenishes both CPU-
and memory-budgets to the maximum values periodically. At
the beginning of each server period its dynamic attributes are
set as $q_s := Q_s$, $m_s := M_s$. In each core, the node scheduler
is responsible to schedule all ready servers and it selects a
highest priority ready server for execution. A server is ready
to execute if it possesses both remaining CPU- and memory-
budgets, formally: $(q_s > 0 \land m_s > 0)$. A higher priority
server can pre-empt the execution of lower priority servers.
During the server’s execution, its CPU-capacity, $q_s$, decreases
with the progression of time, while its memory-bandwidth
capacity, $m_s$, decreases when a task in the server issues a
memory request. A server which depletes any of its resources
is suspended from execution and waits for its replenishment
at the beginning of the next server-period. Thus, if any of the
budgets is depleted then the other remaining budget will be
discarded, i.e., if $m_s = 0$ or $q_s = 0$ then $m_s = q_s = 0$.

The idling periodic server strategy [27] is used for CPU
reservation, i.e., if the scheduled server has remaining budget
but there is no task ready then it simply idles away its CPU-
budget until a task becomes ready or one of the server's
budgets depletes. A scheduled server uses its local scheduler
to select a task to be executed. A higher priority task can pre-
empt the execution of lower priority tasks but not while the
core is stalling. The details of the implementation of MRS and
its execution can be found in [28], [8].

C. Task model

We are considering a simple sporadic task model in which
each task $\tau_i$ is represented as $\tau_i(T_i, C_i, D_i, CM_i)$ where $T_i$
denotes the minimum inter-arrival time of task $\tau_i$ with worst-
case execution time $C_i$ excluding the memory interference
delay, so $C_i$ is the worst case execution time to execute
the task code without including the time to fetch the required data
from memory into the cache. $D_i$ denotes the deadline of the
task where $D_i \leq T_i$. The tasks are indexed in reverse priority
order, i.e. $\tau_i$ has priority higher than that of $\tau_{i+1}$. $CM_i$
denotes the maximum number of cache miss requests and the
time of issuing a cache miss request is arbitrary during
the task’s execution time. Similar to [22], [21] we assume
that each task $\tau_i$ has its own private partition in the cache
that is sufficient to store one row of a DRAM bank. This
assumption can be satisfied by implementing operating system
based cache coloring [29]. Further, we assume that cache-
related preemption delays (CRPD) [30], [31] are zero due to
partitioned cache, and the value of $CM_i$ does not change due
to preemption.

IV. SCHEDULABILITY ANALYSIS

We use the compositional hierarchical schedulability analy-
sis techniques to check the system schedulability by com-
posing the subsystems interfaces which abstract the resource
demands of the subsystems [5]. The analysis is performed
in two levels; the first is called the local schedulability
analysis where for each subsystem its interface parameters are validated locally based on the resource demand of its local tasks. The second level is called the integration or the global schedulability level, where the subsystems interfaces are used to validate the composability of the subsystems.

A. Local schedulability analysis

First, we present the local schedulability analysis without considering the effect of the memory bandwidth part of the multi-resource server, i.e., assuming a simple periodic server, and then we extend the analysis to include the effect of the memory requests. Note that, at this level, the analysis is independent of the type of the server as long as the server follows the periodic model, i.e., both budgets are guaranteed every server period. We assume that the server’s period, CPU-budget, and memory-budget are all given for the local server.

1) Considering only CPU-budget: The local schedulability analysis under FPPS is given by [5]:

\[ \forall t \exists \tau_t : 0 < t \leq D_t, \quad rbf_s(i, t) \leq \text{sbf}_s(t), \]  

where \( \text{sbf}_s(t) \) is the supply bound function that computes the minimum possible CPU supply to \( S_s \) for every time interval length \( t \), and \( rbf_s(i, t) \) denotes the request bound function of a task \( \tau_t \) which computes the maximum cumulative execution requests that could be generated from the time that \( \tau_t \) is released up to time \( t \). \( \text{sbf}_s(t) \) is based on the periodic resource model presented in [5] and is calculated as follows:

\[ \text{sbf}_s(t) = \begin{cases} 
(1) & t = (k-1)(P_s - Q_s) - BD_s, \quad \text{if } t \in W^{(k)} \\
(2) & \text{otherwise},
\end{cases} \]

where \( k = \max \left( \left\lceil \left( \frac{(t + (P_s - Q_s) - BD_s)}{P_s} \right) \right\rceil, 1 \right) \) and \( W^{(k)} \) denotes an interval \( [(k-1)P_s + BD_s, (k-1)P_s + BD_s + Q_s] \). Blackout Duration \( BD \) is the longest time interval that the server cannot provide any CPU resource to its internal tasks and it is computed as \( BD_s = 2(P_s - Q_s) \). The computation of BD guarantees a minimum CPU supply, in which the worst-case budget provision is considered, assuming that tasks are released at the same time when the subsystem budget has depleted, the budget has been served at the beginning of the server period, and the following budget is supplied at the end of the server period due to interference from other higher priority servers.

For the request bound function \( rbf_s(i, t) \) of a task \( \tau_t \), to compute the maximum execution requests up to time \( t \), we assume that \( \tau_t \) and all its higher priority tasks are simultaneously released. \( rbf_s(i, t) \) is calculated as follows:

\[ rbf_s(i, t) = C_i + \sum_{\tau_k \in \text{HP}(i)} \left\lfloor \frac{t}{T_k} \right\rfloor \times C_k, \]

where \( \text{HP}(i) \) is a set of tasks with priority higher than that of \( \tau_t \). Looking at (3), it is clear that \( rbf_s(i, t) \) is a discrete step function that changes its value at certain time points \( t = a \times T_k \) where \( a \) is an integer number. Then for (1), \( t \) can be selected from a finite set of scheduling points \( \{SP_i\} \).

2) Computing worst case delay for a single DRAM memory request: depends upon few characteristics that influence the memory access time of DRAM, i.e., row-conflicts, a change in the data bus direction for each request, and rescheduling algorithm. Row-conflict means that the currently opened row is different than the requested row. In this case, first the opened row is saved and then the requested row is fetched into the row-buffer. A row-conflict consists of three DRAM commands: ACT (activate command loads the requested row into the row-buffer), PRE (precharge command writes back the currently opened row) and CAS (RD/WR) (read/write command reads or writes the required data from/to the row-buffer). Thus \( D\ell \) is a sum of latencies for ACT, PRE and CAS commands plus the DRAM timing constraints to meet these three commands.

When the current command is different than the previous command (e.g. current command is read while the previous was write) then the direction of the data bus needs to be changed. It implies additional timing constraints on the execution of the request, and increases the time to execute the memory request. DRAM controller performs internal scheduling algorithms (i.e. First-Ready First Come First Served) to maximize the overall throughput [25]. Since the row-hit latency is much less than the row-conflict latency, the DRAM controller prefers the row-hit requests over the row-conflict requests. These timing constraints are taken into account into the analysis [23].

\[ D\ell = (Dl_{\text{PRE}} + Dl_{\text{ACT}} + \max(TC_{\text{PRE}}, TC_{\text{ACT}}) + Dl_{\text{CAS}} + TC_{\text{CAS}}) \times t_{\text{CK}} \]

where \( Dl_{\text{PRE}}, Dl_{\text{ACT}}, Dl_{\text{CAS}} \) present latencies (or delays) for ACT, PRE and CAS commands respectively, while \( TC_{\text{PRE}}, TC_{\text{ACT}}, TC_{\text{CAS}} \) present the timing constraints for these commands respectively. The above mentioned characteristics that influence the memory access time of DRAM are taken care into these timing constraints. \( t_{\text{CK}} \) is DRAM clock cycle equals to 1.5 nanoseconds for the DRAM DDR3-1333MHz device.

The latency values for DRAM commands and timing constraints are different for private and shared banks. The analysis for both \( D\ell^p \) and \( D\ell^e \) is presented in [23]. Since only one, either private or shared, bank can be used at a time, we only use the term \( D\ell \) in the analysis. For experiments, we compute values for both \( D\ell^p \) and \( D\ell^e \), and use them accordingly.

3) Considering CPU- and memory-budget: In [15], [16], [22], the effect of the memory bandwidth access has been included in the calculations of the response times of tasks. The basic idea in all these works is the computation of the maximum interference \( MI(t) \) caused by the memory-bandwidth contention on tasks during time interval \( t \). The new request bound function including the memory-bandwidth contention is provided in (5).

\[ rbf^*_s(i, t) = C_i + \sum_{\tau_k \in \text{HP}(i)} \left\lfloor \frac{t}{T_k} \right\rfloor \times C_k + MI(t). \]
MI(t) is computed by multiplying the time needed for a request to be completed by the upper bound of memory-bandwidth requests in t, issued by all the other tasks (executed by all the other servers) located on cores other than the one hosting the analyzed task [22]. However, this method cannot be used in our case since we assume that the subsystems are developed independently hence the tasks’ parameters that belong to the other cores are not known in advance. In addition, the effect of both budgets (CPU- and memory-) should be accounted for in the MRS, which has not been considered in the previous works.

To solve this problem, we focus on the memory-bandwidth requests that can be generated by the tasks running inside a MRS. Considering the behaviour of MRS, we can distinguish two cases that can affect its tasks’ execution.

1) When a task \( \tau_i \), executing in \( S_s \), issues a memory request that causes a miss in a local cache, the associated core is stalling until the cache-line is fetched from memory. The maximum time that the task can be delayed due to the core stalling is presented as \( D_{\ell} \) and this delay should be considered in the analysis.

2) CPU-budget depletion due to memory-budget depletion. When tasks belonging to the same server issue \( M_s \), memory requests, the memory-budget will deplete which will force the CPU-budget to be depleted as well. In the worst case, \( M_s \) memory requests can be issued from tasks of the same server \( S_s \) sequentially, i.e. tasks send a new request directly after serving the current one. If this happens at the beginning of the server execution, a complete CPU-budget will be dropped and the server’s internal tasks will not be able to execute during this server period (this case is shown in the first server period in Figure 2).

\( NR(i, t) \) denotes the maximum number of memory requests that can be generated during a time interval \( t \). During the execution of \( \tau_i \), it can be delayed by at most one memory request sent from a lower priority task and by the number of requests that the task itself sends and finally by the higher priority tasks that can preempt its execution. (6) is used to find \( NR(i, t) \). Note that we include the delay due to a memory request sent from a lower priority task in the equation by adding one in \( CM_i + 1 \).

\[
NR(i, t) = CM_i + 1 + \sum_{\tau_j \in \text{FP}(i)} \left\lceil \frac{t}{T_k} \right\rceil \times CM_k, \tag{6}
\]

Finally, \( MI(t) \) in (5) is calculated using (6)

\[
MI(t) = NR(i, t) \times D_{\ell} \tag{7}
\]

Example 2
\( Ps \)
\( Qs \)
\( Ms \) \( D_{\ell} \)
Legend:
- Processor stalling
- Release time
- Memory access
- Unused CPU budget

Fig. 2: An example illustrating the worst-case CPU-supply

\( Ms \) \( D_{\ell} \)
Legend:
- Memory access
- Unused CPU budget

Fig. 3: New supply bound function \( sbf^*(i,t) \)

Note that the first part of the min function in (8) provides the upper bound of server periods when the CPU-budget can deplete due to the memory-budget depletion during the time interval \( t \). It can be calculated by dividing the total number of memory request generated (by the task \( \tau_i \)) during time interval \( t \) by the memory-budget of the server. However, it cannot exceed the number of server periods up to \( t \), which is bounded in the second part of the function.

\[
A(i, t) = \min \left( \left\lceil \frac{NR(i, t)}{Ms} \right\rceil, \left\lceil \frac{t}{Ps} \right\rceil - 1 \right) \tag{8}
\]

By computing \( A(i, t) \), we can consider the effect of the memory budget part on the \( sbf_\tau(i,t) \) by assuming that \( A(i, t) \) CPU budgets will not be provided up to \( t \), i.e., the server budget \( Q_s = 0 \) whenever memory budget depletes. This can be achieved by increasing \( BD \) in (2) by \( A(i, t) \times Ps \) which is equivalent to removing \( A(i, t) \) CPU budgets, i.e., \( A(i, t) \times Q_s \) from the supply bound function. However, the CPU budget will be depleted due to the depletion of memory budget only after serving \( M_s \) requests which is equivalent...
to providing $M_s \times D\ell$ CPU resource every server period as shown in Figure 3 (remember that each memory request delay $D\ell$ is modeled as an extra CPU demand in the $rbf_s^*(i,t)$). To decrease the pessimism in the analysis, we assume that $A(i,t) \times M_s \times D\ell$ will be added in the calculation of $sbf_s(t)$ which makes this function different for different tasks. However, it will be correct only if $M_s \times D\ell < Q_s$. The supply bound function $sbf_s^*(i,t)$ for $\tau_i$ is computed as follows.

$$sbf_s^*(i,t) = \begin{cases} t - (k(i,t) - 1)(P_s - Q_s) - BD_s(i,t) + A(i,t) \times M_s \times D\ell & \text{if } t \in W^{(k)}(i,t) \\ +A(i,t) \times M_s \times D\ell & \text{otherwise} \end{cases}$$

(9)

where $k(i,t) = \max \left( \left( t + (P_s - Q_s) - BD_s(i,t) \right) / P_s, 1 \right)$ and $W^{(k)}(i,t)$ denotes an interval: 

$$[(k(i,t) - 1)P_s + BD_s(i,t), (k(i,t) - 1)P_s + BD_s(i,t) + Q_s]$$

and $BD_s(i,t) = 2P_s - Q_s + A(i,t) \times P_s$.

B. System integration

During the integration phase of MRSes, all servers should be guaranteed to receive the required CPU and memory budgets specified in their interfaces. To validate this, two different tests should be applied. The first test is performed on the CPU part to make sure that the required CPU budget will be provided. The second test is performed to make sure that the total memory bandwidth usage by all servers in the system is lower than the maximum available bandwidth of the memory bus. Both tests can be performed independently and should succeed to guarantee that all tasks meet their deadlines. Therefore the parameters that are provided in the interface of each subsystem $S_i$ to apply both tests are $P_s, Q_s, M_s, D\ell$.

As described earlier, the value of $D\ell$ depends on the hardware architecture. This keeps our local analysis independent of other servers in the system. As a simple example and assuming the FR-FCFS policy and knowing that only one request can be sent from each core at a time (since a core is stalling when a request is sent), then the upper bound value of $D\ell$ equals to the number of cores multiplied by the time taken to serve each request, plus adding the reordering effect of FR-FCFS policy, as presented in [23]. The reason is that for each core when it tries to send a memory request, as a worst case all other cores send one request just before the core under analysis, and one request from a lower priority task on executing on the same core, which bounds the number of requests.

**Global schedulability test for CPU-budget:** Since the CPU part of the server is of periodic type, each subsystem can be modeled as a simple periodic task where the subsystem period is equivalent to the task period and the subsystem budget is equivalent to the worst case execution time of a task. Then the schedulability analysis used for simple periodic tasks can be applied on all servers that share the same core for this test [5]. $R_i^{k+1} = Q_i + B_i + \sum_{j \in \text{REP}(i)} \left( \frac{P_j}{T_j} \right) \times Q_j$. The test starts with $R_i^0 = Q_i$. The test is stopped when $R_i^{k+1} = P_i$ and $R_i^{k+1} \leq P_i$. If $R_i^{k+1} > P_i$, then system is not schedulable. Note that since for each memory request, the associated core is stalling then a higher priority server may be blocked by a lower priority server at most once with maximum blocking time equals to $B_i = D\ell$. This blocking time is considered in the analysis.

**Global schedulability test for memory-budget:** The maximum memory-bus bandwidth used by all servers on all cores (denoted by $B_{max}$) should be less than the minimum bandwidth of the memory-bus. The practical minimum bus bandwidth rate ($B_{avail}$) that can be used to access data from DRAM has some practical limits and is less than the maximum bandwidth of the bus. It is difficult to obtain this bound from documentation, therefore, it is experimentally measured. The practical minimum bus bandwidth rate measured for Intel Core2Quad Q8400 processor is $B_{avail} = 1198 MB/s$ [32]. We experimentally measured it $B_{avail} = 1022 MB/s$ for our Intel core 2 CPU 6700 architecture.

For global schedulability test, the maximum bandwidth ($B_{max}$) used by all servers on all cores should not exceed this limit $B_{max} \leq B_{avail}$. As the memory-bus is shared among all cores, therefore, we sum up all requests from all servers from all cores. $B_{max}$ is computed as $B_{max} = \sum S_i \left( \frac{M_s}{P_s} \times 64 \times 1000 / (1024 \times 1024) \right)$. The number of memory requests are converted to the bandwidth by dividing with server period $P_s$, multiplying it with the size of cache line (i.e. 64 bytes). To convert the service rate to MB/s, it is multiplied with 1000 and divided by $(1024 \times 1024)$. The server period is given in $ms$, that's why it is multiplied with 1000.

V. INVESTIGATING CPU- AND MEMORY-BUDGETS

In this section we investigate the relationship of CPU- and memory-budgets and the effect of increase/decrease of memory-budget $M_s$ on CPU-budget $Q_s$ of a MRS $S_i$ using synthetic experiments. We look into the effects of private and shared memory banks on the server.

A. Evaluation setup

We consider multicore system using quad-processors, and DDR3-DRAM 1333H memory controller with 8 banks per rank. COTS architectures with these specifications are available (e.g. Intel Core i5). The upper bound of $D\ell$ is computed for both private ($D\ell^p$) and shared banks ($D\ell^s$) in nano seconds (ns) and the value of $D\ell^s$ is almost double than that of $D\ell^p$.

1) Two different task behaviours: Two different synthetic task-types are used in our synthetic evaluations, namely normal task, and memory intensive task. The normal task generates a relatively low number of memory requests ($CM_i = 1000$) per task period as compared to the memory intensive task. The memory intensive task generates higher number of memory requests ($CM_i = 20000$) per task period. Thus this task will heavily affect the memory budget requirements of the server and will also effect the CPU-budget of the associated MRS indirectly.
2) Timing properties of a MRS and its task set: Since we have previously shown the composition of MRSes in [8], in this paper we focus on the individual behaviour of a single server and how a server’s parameters are affected from its tasks.

A single MRS is considered for the experiments with a period of 60 ms, and consisting of three tasks: two normal tasks and one memory-intensive task. A normal task generates 1000 memory requests per task period, while a memory-intensive task generates 20000 requests per task period. The timing properties of the three tasks are presented in Table I.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>τ₁</th>
<th>τ₂</th>
<th>τ₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>H</td>
<td>M</td>
<td>L</td>
</tr>
<tr>
<td>Period (ms)</td>
<td>100</td>
<td>320</td>
<td>640</td>
</tr>
<tr>
<td>WCET (ms)</td>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>CM</td>
<td>1000</td>
<td>1000</td>
<td>20000</td>
</tr>
</tbody>
</table>

Table I: Task properties.

3) Calculating minimum and maximum memory-budgets: We assume that the server period is given similar to [5], which is required to evaluate both CPU and memory budgets. We first calculate a range of minimum and maximum values for the memory-budget, and then for each value within the range, we evaluate the minimum CPU-budget so that the system remains schedulable using equations 1, 5, 9. The range, we evaluate the minimum CPU-budget so that the memory-budget of the system remains schedulable using equations 1, 5, 9. The minimum and maximum bounds for the memory-budget of the server respectively. From the memory perspective, each task should be able to issue all its memory requests CMᵢ within its period Tᵢ and its server should serve these requests until the deadline. Thus Mᵢ = max ∀ᵢ (CMᵢ/ [Tᵢ−Pᵢ]), and Mₛ ≥ Mᵢ condition should be satisfied. Mₛ is computed as Mₛ = max ∣Nᵢ(R(i, Tᵢ))−sbfᵢ(i, Tᵢ)∣, where Nᵢ(R(i, Tᵢ)) represent the maximum number of generated requests till the deadline of task τᵢ, and for Mₛ max we consider that all these requests are generated in one server period Mₛ. More than these requests cannot be generated during this period.

B. Synthetic experiments

The main focus of performing synthetic experiments is to investigate the behaviour of MRS by changing different parameters, like: (1) changing the value of memory-budget and exploring its effect on the CPU-budget’s value; (2) checking the effect of private and shared memory banks; (3) changing the priority of memory-intensive tasks and investigating its effect on both budgets; and (4) increasing memory request of a task and observing its effect on both budgets of the server. Mᵢ and Mₛ values are computed and the experiments are conducted for that range of Mₛ. The upper bound values for Dₛ and Dₛ are used for private and shared memory banks respectively.

1) Experiment 1: Correlation between CPU- and memory-budgets: The purpose of this experiment is to investigate the correlation between both budgets Mₛ and Qₛ. The timing properties of the MRS and its tasks set used for this experiment are presented in Table I. The results are presented in Figure 4 where the x-axis denotes the range of Mₛ, and the y-axis shows the minimum CPU-budget Qₛ for which the server is schedulable. Note that for better presentation of the graph, we shortened the shown range of Mₛ values.

![Graph showing the correlation between Mₛ and Qₛ](image)

This graph shows a stair-function, the value of Qₛ decreases at certain points with the increase of Mₛ. The reason is that when the value of Mₛ is minimum, and memory requests are generated at the start of the server period, then Mₛ depletes after Mₛ × Dₛ time and all the remaining CPU-budget for that period is simply discarded (as shown in second and third server period in Figure 3). The demand of Qₛ is high as more server periods are needed to execute the tasks. Conversely, the increase of Mₛ decreases the value of A(i, t) in equation 8. This results in a decrease BDₛ(i, t) (see equation 9), and an increase in sfᵢᵢ(i, t), thus requiring less Qₛ.

2) The Effect of private and shared memory banks: Figure 4 presents the results of using both private and shared memory banks using Dₛ and Dₛ respectively. We note in this experiment that the choice of private or shared banks does not affect the needed CPU-allocation, Qₛ, much. Often the needed allocation is the same regardless which memory organization is used. And in the rather few cases when private banks allow a smaller allocation of Qₛ, the decrease in Qₛ is negligible. It is mainly due to a big difference between time unit of memory-interference delay (nano sec) and the time unit of server period and CPU-budget (ms).

3) Experiment 2: Impact of the period of memory-intensive task on server budgets: We performed this experiment by changing the number of memory requests of the tasks in the previous experiment, i.e., first the high priority task τ₁ in Table I generates 20K requests, τ₂ and τ₃ generate 1000 requests each. It means that the period of memory-intensive task is 160. Second, the medium priority task τ₂ generates 20K (i.e. the period of memory-intensive task is 320) and τ₁ and τ₃ generate 1000 requests each. Third, the low priority task τ₃ generates 20K other tasks generate 1000 requests (i.e. the period of memory-intensive task is 640 now). Other properties of tasks remain the same as presented in Table I.

We see in Figure 5, the need for memory-budget increases a lot when the memory-intensive task is executed with a higher priority. There are two reasons for this effect. First, as
rate monotonic is used for priority assignment, therefore, the shorter period task is activated more often than other longer period tasks, that leads to an increase in the total number of generated requests during a time interval \( t \). It increases the memory-budget \( M_s \) of the server period (obvious from the graph of \( \tau = 20K \) in Figure 5 where MRS is schedulable for a higher \( M_s \) value). The increase in the number of requests increases the value of \( A(i,t) \) in (8), consequently \( BD_s(i,t) \) increases as well (see eq. 9). Second, the higher priority task affects the request bound function of all the lower priority tasks by adding the memory interference delay \( MI(t) \) to their \( rbf_s \) (see eq. 5), thus increasing their \( rbf_s \). If the memory-intensive task has lowest priority, then its \( MI(t) \) does not effect other tasks in the server.

4) Experiment 3: Impact of different server periods on server-budgets: This experiment is performed for different server periods (ranging from 20ms till 80ms) for the task set presented in Table I. The results are presented in Figure 6, where x-axis presents the server’s memory-bandwidth utilization \( (M_s/P_s \times 64 \times 1000/(1024 \times 1024)) \) in MB/s), and y-axis presents CPU utilization\% \( (Q_s/P_s \times 100) \).

In our results, sometimes the longer server period has a smaller CPU utilization when memory utilization is small as compared to the shorter server periods (i.e. in Figure 6, \( P_s = 80ms \) has smaller CPU utilization at point 2.5MB/s than for \( P_s = 40ms \), and for \( P_s = 20ms \) at point 2.5MB/s the system needs more than 100% CPU utilization and is thus not schedulable). However, when the memory utilization is increased, the shorter server periods need a smaller CPU utilization.

In general using traditional server-based scheduling, a longer server period results in an increase in the blackout duration that requires bigger CPU budget to schedule the server [5]. However, from the memory perspective of the MRS, looking at equation 8, the increase of memory budget has a big impact on \( A(i,t) \) function. Bigger value of \( M_s \), due to the floor function, decreases the value of \( A(i,t) \) in equation 8, which in turn decreases the CPU budget requirement to schedule the server. We observe in Figure 6 that at smaller values of memory utilization (i.e. 2.5MB/s), the longer server period \( (P_s = 80ms) \) brings smaller CPU utilization because the impact of equation 8 dominates. In other cases where memory utilization has increased, the blackout duration effect is dominating. Thus we conclude that the behaviour of MRS differs from the behaviour of traditional servers.

5) Candidate interfaces for the MRS: An interface of a MRS specifies the timing properties of the subsystem precisely. A candidate interface represents the parameters from some range of valid interface parameters that increase system composability while simultaneously ensuring subsystem schedulability. For a MRS, the problem is to determine both parameters (CPU-bandwidth and memory-bandwidth) while keeping the server period fixed (usually half of its shortest task period [5]).

We present an example consisting of four servers executing on two different cores with task sets presented in Table II.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>( T_0 )</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
<th>( T_5 )</th>
<th>( T_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>50</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>52</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>Prio</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>T</td>
<td>40</td>
<td>80</td>
<td>160</td>
<td>80</td>
<td>160</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>WCET</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>10</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>CM</td>
<td>20K</td>
<td>40K</td>
<td>80K</td>
<td>40K</td>
<td>60K</td>
<td>60K</td>
<td>60K</td>
</tr>
</tbody>
</table>

**TABLE II:** Task properties.
6) Discussion: From experiment 1, we observe that $Q_s$ decreases significantly at the start with the increase of memory budget $M_s$. And after a certain value, more increase in $M_s$ does not affect the value of $Q_s$ much. This helps in selecting the suitable values for $Q_s$ and $M_s$. We also observe (see Exp. 3) that the behaviour of MRS is not similar to the traditional server. The change in server period has different impacts on CPU and memory budgets.

The presented analysis in this paper is pessimistic. We can identify some factors that contribute to the pessimism. The biggest reason of pessimism is our consideration of worst case $M_s$ depletion where all memory requests are generated at the start of the server period. As a result $M_s$ depletes and the remaining $Q_s$ is discarded (see Figure 3). However, it could not be the case in reality. To improve the analysis, we should look at the start of task periods. We leave this improvement as a future work.

Over-provisioning the budget: typically in server based scheduling, if possible, we can over-provision the resource to get the shorter response times of the tasks. We calculated $Q_s$ without considering memory requests ($CM_i$ is 0 for all tasks). We get a constant value for $Q_s$ which can be considered as a reference point. By adding memory requests $CM_i$, the value of $Q_s$ will increase slightly. In order to get the shorter response times for the tasks, the value of $Q_s$ can be increased (budgets can be over-provisioned), as long as the global schedulability of the system is satisfied.

VI. CONCLUSION

We have proposed a multi-resource server (MRS) approach to address composability of independently developed real-time subsystems executed on a multicore platform. The memory-bandwidth is added as an additional server-resource to bound memory interference from other servers executing concurrently on other cores thus to provide predictable performance of multiple subsystems. Consequently, tasks within a multi-resource server execute provided with both CPU- and memory-budgets. In this paper, we have presented a compositional analysis framework for MRS including a complete and composable local and global analysis. For memory interference, we have safely bounded the memory contention for DDR DRAM memory controller that are commonly used in COTS multicore architectures. Further, we have performed an experimental study to investigate the relationship between the server parameters including memory-budget and CPU-budget of the server and gives indications to evaluate the optimal parameters. Finding optimal interfaces for the MRS is an open issue.

We have explored the source of pessimism in our analysis and in future we intend to improve the analysis. It would be interesting to find an algorithm to calculate the optimum budgets for both resources of the MRS and to find smart online algorithms to assign the unused capacity of one resource to another server to improve overall average response times. Another direction is to transcend the boundaries of software engineering and real-time systems by embedding the MRS

![Fig. 5: Effect of High- and low-priority memory-intensive task on $M_s$ and $Q_s$ using private banks only](image-url)
within the software components [33], [34] running on multicore platforms.

REFERENCES


[24] Micron. 2Gb DDR3 SDRAM.


