

High-Level Dynamic Resource Management for Distributed, Real-Time Embedded Systems

Kurt Rohloff, Richard Schantz
{krohloff, schantz}@bbn.com
BBN Technologies
10 Moulton St.
Cambridge, MA, USA

Yarom Gabay
yarom@cs.bu.edu
Department of Computer Science
Boston University
111 Cummington St.
Boston, MA. USA

Abstract*

In this paper we discuss the problem of coordinating resource allocations among independent high-level goals, called missions, for scalable, hierarchical, Dynamic Resource Management (DRM) in a Distributed Real-time Embedded (DRE) system. The DRM goal is to dynamically allocate shared resources to simultaneously manage multiple Quality of Service (QoS) concerns among the missions that maintain system operation despite partial system failures. We use a utility-driven approach for decision-making and performance evaluation. We offer an approach for multi-mission coordination and provide a design for implementing that approach. Finally, we show experimental results demonstrating the viability and near-optimality of our solution for a target environment based on a large-scale Matlab/Simulink simulation of a target system.

1. Introduction

Distributed Real-time and Embedded (DRE) systems are becoming increasingly common and important as the underlying technology for distributed computing and networking systems continues to mature. Examples of modern DRE systems include military command and control systems, manufacturing process automation and mission-critical audio/video systems. DRE systems are often designed with *static* and *ad hoc* resource-allocation strategies inappropriate for these potentially agile and potent systems. One of the greatest challenges in the design of DRE systems is the necessity of effective and scalable Dynamic Resource Management (DRM). The goal of DRM is to adapt a system's resource allocation to dynamically changing conditions, such as changing mission priorities and resource

failures, to maximize system performance. Other approaches in the literature to adaptive software and resource systems for QoS optimization include [1, 3, 4, 8, 9].

We previously presented an initial design for a scalable hierarchical utility-driven DRM architecture for a large scale DRE system [6]. This DRM architecture relies on the decomposition of resource management into multiple levels of abstraction for a more scalable approach to very large systems. In this paradigm we break the behavior of the system into a behavioral hierarchy where the overall high-level system goals are broken into sub-goals which we call *missions*. These missions are further broken into strings of software applications which are deployed onto the system's resources. A schematic of the decomposition of system behavior can be seen in Figure 1.

In our resource coordination paradigm, missions are not allocated specific resource, but the rights to use amounts of resources so that missions can independently choose which resource they use while not being limited to using specific resources. In this paper we focus on the problem of the highest system-level resource coordination where missions are allocated the rights to use specific amounts of resources.

This work is inspired by the DARPA Adaptive and Reflective Middleware Systems (ARMS) program. The

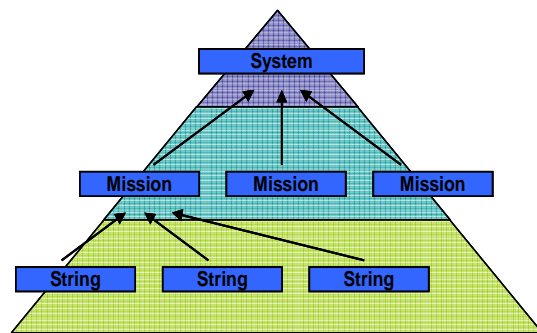


Figure 1: Example Behavioral Decomposition of DRE Concept System

* This work was supported by the Defense Research Projects Agency (DARPA) under contract NBCHC030119.

ARMS program is aimed at developing a new generation of capabilities for the dynamic allocation of real-time applications for a concept system which needs to satisfy multiple, possibly conflicting behavioral requirements.

A key aspect of our DRM resource usage concept is a composite utility function to measure system performance at multiple levels of abstraction. The utility function is defined at each level of the resource hierarchy to reflect local performance with respect to Quality of Service (QoS) requirements. Resource managers, called *controllers*, are deployed at the various levels of the hierarchy, mirroring the decomposition of the system. These controllers take actions to greedily increase their local utility while allowing higher-level managers to restrict and guide the actions of subordinate controllers through policies.

The mission-level controllers, known as *Mission Controllers* (MCs), work independently of one another to acquire global resources from the pool of available resource that they have been granted rights to use for the deployment of their software applications. These resource usage rights take the form of resource-consumption levels which the MCs must not exceed. During operation, the MCs acquire resources in order to maintain their mission utility, while not exceeding their assigned resource usage level.

The simultaneous operation of multiple independent MCs will not conflict as long as the amount of resources available is sufficient for the deployment of all the software applications in the system. However, under conditions of limited, changing resource availability, the system can exhibit an inefficient and erratic division of resources and thus low or unacceptable utility. When a resource failure occurs, MCs may not have sufficient resource rights to redeploy important applications in the system from the failed resource to operational resources. Ideally, the MCs should adapt to the failure by decreasing their resource consumptions in order to ideally achieve as high system utility as possible. Due to computational complexity issues, it is almost always the case that it is infeasible to find a resource allocation that would achieve the optimal utility in a reasonable amount of time for a real-world system, so we focus on heuristic distributed methods to find resource allocations with as high utility as possible.

Because the MCs are operating independently, they are unaware of updated resource availability levels after resource failures. Instead, after resource failures, the MCs attempt to reacquire lost resources for their failed strings in order to increase mission utility back to its original level by redeployed their failed application strings onto operating resources. Consequentially, if there is no coordination between the missions, resources that should be allocated to more important

applications for some missions may be used by less important strings belonging to other missions. This leads to poor system utility.

We present the design for a general *Multi-Mission Coordinator* (MMC) that dynamically reconfigures resource consumption levels among the missions based on predictive utility computations. MMC operation is driven by system events such as partial system failures or changes in mission priorities.

Ideally, the MMC would at all times have complete information about mission resource status before assigning resource usage allocations to the missions. However, this approach for MMC design is impractical. Not only is communicating the dynamic mission resource states to the MMC generally infeasible due to limited bandwidth, but the problem of optimally allocating resources using this information is computationally difficult as previously indicated.

We offer a heuristic MMC design in which the MCs send *mission tables* to the MMC that contain quantized information about the resource requirements corresponding to levels of utility that could be provided by the missions. The choice of quantization levels in the tables are a tradeoff between the accuracy of information about resources requested by the missions to provide various levels of quality of service and the communication overhead associated with transmitting this information.

We present our MMC concept of operations which computes the gross-level allocations of resource to the mission based on the mission tables and known resources. We take a dynamic programming approach for MMC operation which computes an optimal high-level resource allocation with respect to the quantized (and hence approximate) mission tables provided as input to the MMC.

We demonstrate the viability of our solution by giving experimental results from a simulated system in response to partial resource failures. We present a large-scale Matlab/Simulink simulation of a representative DRE system for which we computed optimal resource allocations offline for various levels of resource availability. We show that for this representative system, our DRM system selects resource allocations with utility levels nearly as high as the optimal resource allocation for many resource allocation levels. Additionally, our DRM system always performs better than a baseline system currently used in our target representative system.

The rest of the paper is organized as follows. In Section 2, we present the DRE system. In Section 3 we present the DRM architecture along general lines. In Section 4, we discuss the MMC architectural and algorithmic design. In Section 5, we present the simulation experiment results. Finally, we discuss future directions in Section 6.

2. The DRE System

Properties of DRE systems can be understood via aspects of both their resources and applications running on those resources. The resource aspects of DRE systems include a set of general-purpose computational nodes and communication links which connect the nodes. Nodes are assumed to be grouped into pools or clusters of computing resources based on their physical locations. Communication between nodes in the same pool is assumed to be inexpensive, while nodes in a pool share more limited communication gateways to nodes in other pools. Therefore, communication between nodes is partitioned into intra- and inter-pool communications. A schematic of the system's physical resources can be seen in Figure 2.

Software applications are deployed onto the computational resources and can be viewed at multiple levels of abstraction. At the lowest level of abstraction, *applications* run on nodes and perform work requiring certain computing resources. At the next level of abstraction, an *application string*, or string, is a logical sequence of applications that sequentially process information with unique starting and ending applications. A string is considered deployed if each of its applications and communication links has sufficient resources to operate properly. Strings can be generally deployed across multiple nodes and pools. A mission is a group of strings that cooperate to achieve common goals. At the highest level of abstraction, the system incorporates all running missions and resources to which those missions have access.

Strings and missions are assigned importance values by the users. Through these values, users can express the relative importance of various end-to-end QoS requirements which are used by the DRM controllers to influence resource allocation decisions.

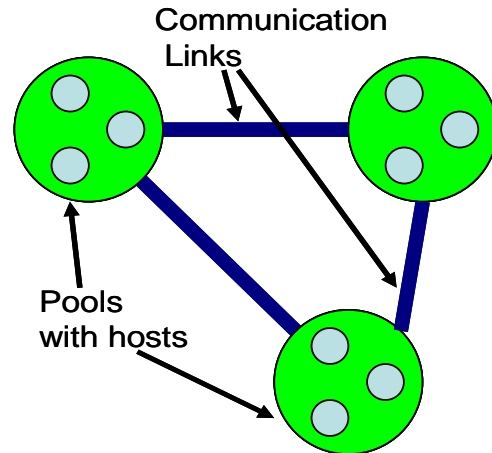


Figure 2: Resources of DRE Systems

Users can choose to dynamically change the importance values during run-time to change the way the system's resources are allocated and hence the results produced.

3. The DRM Architecture

Our DRM architecture mirrors the hierarchical decomposition of the system. A similarly structured composite utility function is used to evaluate the performance of strings and missions in the system and the overall system performance. The utility function is also used to estimate the desirability of various control actions with respect to the future performance and utility of the system. Naturally, the controllers choose control actions that they predict would increase their utility as much as possible.

From our previous work on utility measures for real-time distributed resource allocations [5], we define the system utility as the sum of all mission utilities weighted by the mission importance values. Mission

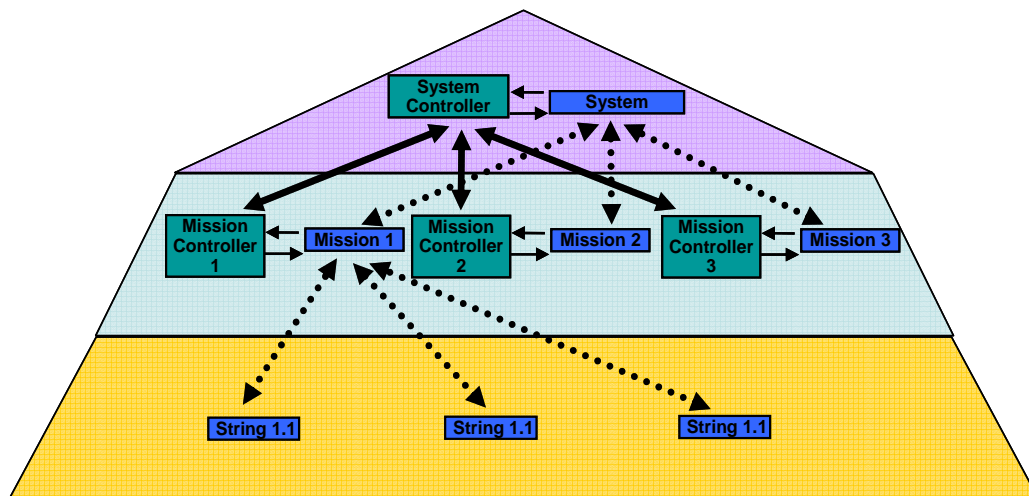


Figure 3: Control Hierarchy of the DRE System

utility is defined as the sum of the importance values of the mission's deployed strings. For this application of our utility measures, we simplify the definition of string utility to be a user-assigned string importance value if the string is assigned sufficient resources to operate, and 0 otherwise.

Controllers are deployed for each software application abstraction to locally monitor and manage resources. Every deployed string has a *String Controller* (SC) that tunes and controls its resource usage. In addition, there is a *Mission Controller* (MC) deployed for every mission in the system. An MC dynamically controls the deployment of strings for the mission to maximize mission utility. For our hierarchical DRM, we use a single system-wide controller, called the *Multi-Mission Coordinator* (MMC), to coordinate gross-level resource allocations among the multiple missions. In this paper we assume a system where the string controllers are unnecessary and not considered. A schematic of this system can be seen in Figure 3.

The controllers respond to events such as decreases in their local utility, usually caused by a failing resource or significant changes in load attributes, as well as allocation policy commands given by higher-level controllers. The general philosophy of the DRM is a bottom-up approach to control actuation. The lower-level controllers are generally fast and responsive, while the higher-level controllers take more invasive and sporadic actions to make large-scale alterations to the resource allocations. In the case of a resource failure, lower-level controllers first try to recover from a utility drop by redeploying failed strings, possibly by using different resources. If they fail to do so the higher-level controllers take more aggressive actions to

recover a more desirable utility level.

Along the lines of the bottom-up approach, scalability is made possible by the localization of information. Controllers at the lower levels have much more detailed and accurate information about the system applications, while higher-level controllers have a more abstract and delayed view of the applications to limit unnecessary information flow in the system.

4. Multi-Mission Coordinator

The MMC manages the gross-level allocation of resources among the missions. It is event driven and operates on the occurrence of major system events: initialization, partial system failures and importance reevaluation. It uses the aggregate level of resources available in the system to compute gross-level resource usage rights to the missions. The MMC communicates these allocations to the MCs so that mission controllers can update their resource profile due to their deployed strings. This high-level multi-mission coordination can be viewed as an optimization problem where the inputs are the available resources in the system and information about the utility achieved by the missions if provided various levels of resource access. The output of this high-level multi-mission coordination is a gross-level allocation of resources usage rights for each mission.

An optimal solution to the multi-mission coordination problem would be attained by considering all the information about all the missions and choosing a set of deployed strings for every mission that would maximize system utility. This approach is infeasible in large-scale systems as this problem is NP-complete. Moreover, communicating the required information

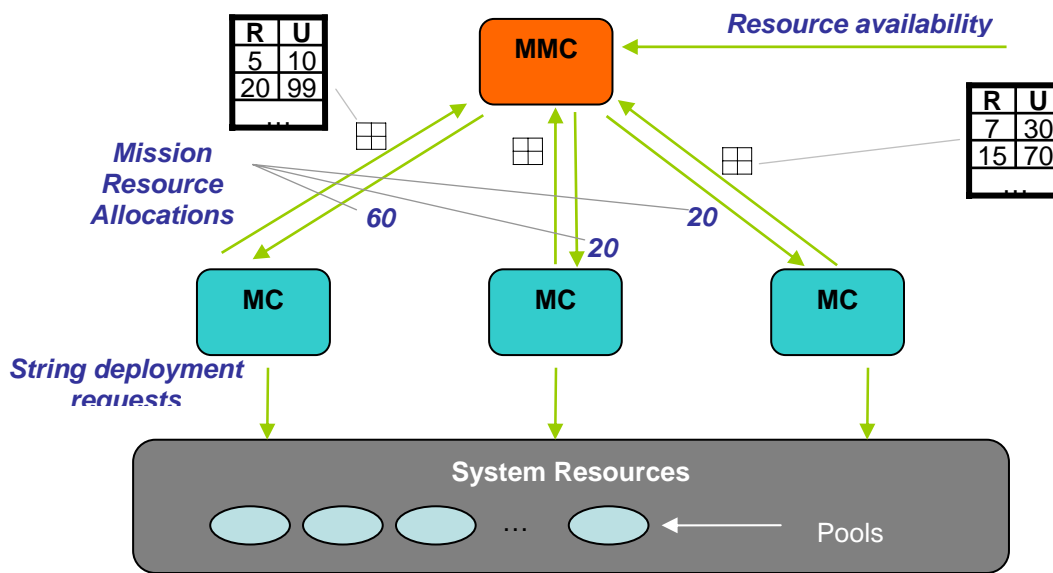


Figure 4: MMC Concept of Operations

from the MCs to the MMC generally requires high communication overhead. Frequently, wide-scale reallocations of resources are performed during times of high resource instability when communication resources are most scarce.

To design a feasible solution to the high-level resource allocation problem, we consider a heuristic approach where we restricted the MMC to computing the gross-level allocations when only a set of mission tables are communicated by every MC to the MMC. The mission tables provide information about how much utility the missions could provide if given access to quantized amounts of resources. These tables essentially allow the MMC to choose missions' deployment levels without having to consider all possible information about the missions. A schematic of the interaction between strings in the MMC and the MCs is given in Figure 4.

Each row in the mission table specifies a level of resource usage paired with the corresponding estimated mission utility. These two values encapsulate information about a set of selected strings along with their estimated utility. One of the rows in the table corresponds to the deployment of the entire mission (equivalently, all strings in the mission.) The resource usage level of this group of strings is the highest level of resource usage that the mission uses. It is possible that a row in the table is used to represent the deployment of only strings critical to the operation of the mission. The resource usage level of this group of strings is the lowest level allowed for this mission; if the mission used any less resources, it would not be able to provide sufficient resources for the operation of its critical strings.

The tables are generated by the MCs at initialization and regenerated every time a mission receives a user command directive to refine its local behavior based on the relative importance value of the mission and its strings. The tables are constructed by considering application-specific information about the mission.

Each row in a table implies a set of strings that would be deployed if the MC is given access to the resources specified by that row in the table. Therefore, the table can account for dependencies between mission strings when strings share applications. In addition, an offline process to select the set of strings that correspond to each resource level in the table can be used to pre-compute the tables used at initialization.

Given the mission tables and information about the availability of resources in the system, the MMC needs to decide how much of each resource to allow each mission to use under current conditions in order to achieve high utility while allocating sufficient resources for the missions' critical strings. In our heuristic, the MMC chooses one row from each mission table so that

the sum of the mission utilities in the selected rows is maximized with respect to the limitation that the sum of the resource usages in the selected rows does not exceed the amount of available resources in the system.

This problem of MMC resource assignment can be formalized as the *multiple-choice knapsack problem* (MK) [2]. Informally, the MK problem is stated as choosing exactly one item from each one of m classes of items such that the total size of the chosen items does not exceed a given capacity and their total value is maximized. Formally speaking, there are n items partitioned into m classes N_k , $k = 1, K, m$. Every item has a value v , where $v_{k,j}$ is the value of the j^{th} item in the k^{th} class. In addition, every item has a size s , where $s_{k,j}$ is the size of the j^{th} item in the k^{th} class. C is the given capacity of the knapsack. There are n binary variables $x_{k,j}$, where $x_{k,j} = 1$ iff the j^{th} item in the k^{th} class is chosen. The problem is to assign values to the variables $x_{k,j}$ such that

$$\text{Maximize: } \sum_{k=1}^m \sum_{j \in N_k} v_{k,j} x_{k,j}$$

$$\text{Subject to: } \sum_{k=1}^m \sum_{j \in N_k} s_{k,j} x_{k,j} \leq C, \text{ where for}$$

$$k = 1, K, m, \sum_{j \in N_k} x_{k,j} = 1 \text{ and } x_{k,j} \in \{0, 1\}$$

Generally speaking, the MK problem is NP-complete as the (regular) knapsack problem is a special case of it. A dynamic programming approach, as described in [2], can be applied in order to compute the maximal solution to the MK problem. The use of dynamic programming for MK relies on the fact that an MK problem can be recursively defined in terms of an MK subproblem. Assuming $MK_k(y)$ denotes the optimal value for a problem with capacity of y and k classes, its recursive definition is as follows

$$MK_k(y) = \max_{j \in N_k} \{v_{k,j} + MK_{k-1}(y - s_{k,j})\}$$

The expression $MK_m(C)$ can then be solved by computing $MK_k(y)$ for $k = 1, K, m$ and $y = 1, K, C$. Although this algorithm runs in pseudo-polynomial time $O(nC)$, we found it to be sufficiently effective and efficient, taking on the order of milliseconds to run on a standard desktop computer for a system with 10's of missions and missions with 10's of entries in their MMC tables.

5. Simulation and Experiment Results

We developed a large-scale, highly configurable Matlab/Simulink model of the multi-mission system in the context of the DARPA ARMS program to objectively compare the utility-measured performance of the system using our dynamic MMC design, a globally optimal resource allocation set precomputed for a selection of resource availabilities and an MMC baseline system where gross-level resource amounts are statically allocated at initialization without further adjustment. The baseline system is representative of current resource allocation system used in the target environment of the ARMS system. In general, it is infeasible to precomputed resource allocations for all possible system configurations that may be required for the system during normal operation, so this is not a valid approach for a deployable MMC methodology.

For our simulation experiments, we configured the model to consist of three missions with 100 strings each that can be deployed on 5 pools with inter-pool communication link resources. When the MCs perform string deployment operations, there is a configurable actuation delay between the time the MC sends the actuation signal until the time the string becomes operational which we approximated as 0.1sec.

In the simulation model, the operating conditions of the strings are highly configurable. The computational and communication requirements of the

strings can be customized to model various mission scenarios. The user-assigned importance values of the strings are also configurable at run-time along with the quantized resource requirement tables that are sent from the MCs to the MMC.

Using the large-scale Matlab/Simulink model of the ARMS system, we generated 100 experimental string deployment scenarios for each of the 3 missions. Each string was assigned randomly chosen application lengths uniformly distributed between 2 and 11. Inter-application bandwidth requirements were randomly chosen to be either 1 or 2 megabits per second. The 100 strings were randomly assigned integer importance values with a uniform random distribution between 1 and 30, inclusive. To generate the mission tables, we randomly grouped the missions' string sets into the 10 quantization levels of the MMC tables where each string has a uniform probability of being assigned to one of the 10 quantization levels.

We simulate partial system failures in real-time in the model by removing all of a pool's nodes to model a complete pool failures. We ran multiple simulation scenarios with our Matlab/Simulink model. For each scenario, the system had five operational pools at initialization with sufficient resources to deploy all strings. The pools were allocated computation resources such that after the failure of a single pool, the system would have only 80% of the resources required

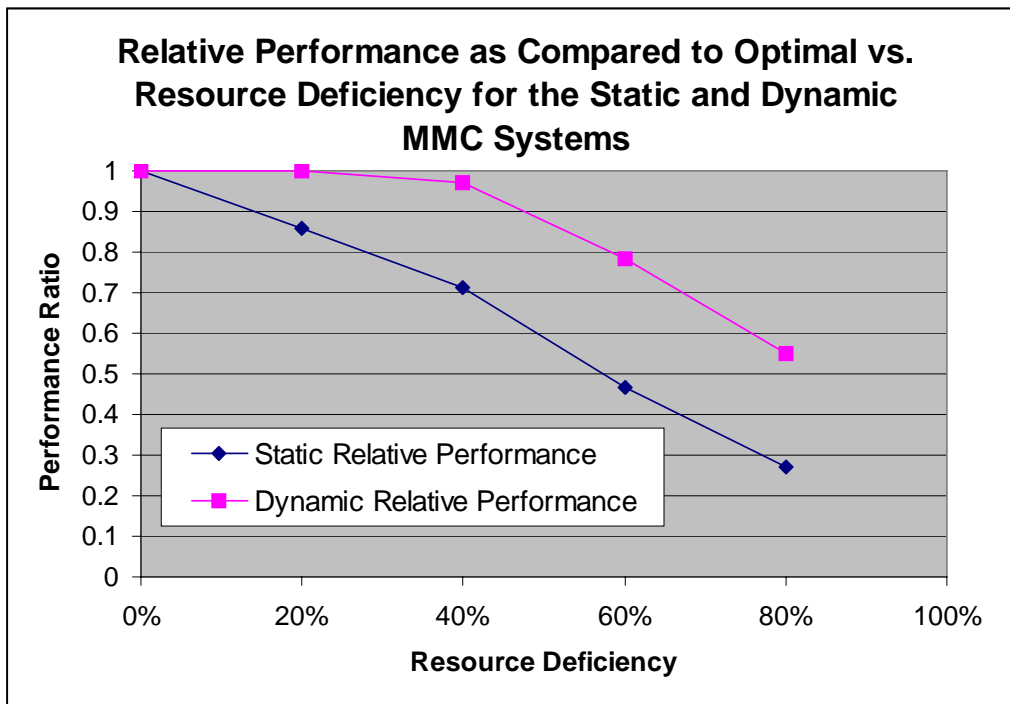


Figure 5: Ratio of Utility Recovery for Dynamic and Static MMC System after Failure with respect to Optimal System.

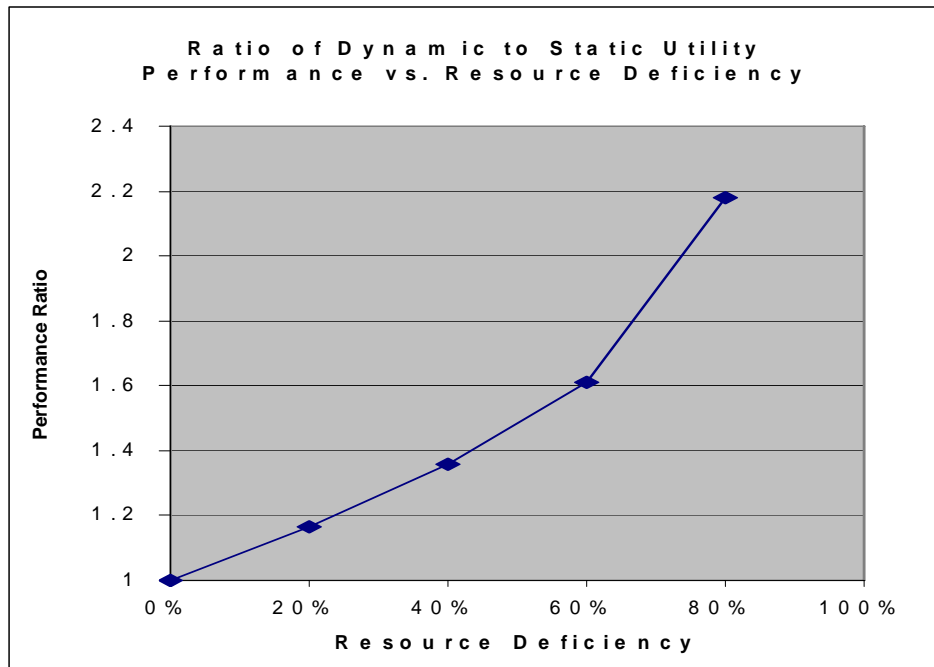


Figure 6: Ratio of Utility Recovery after Failure

to deploy all strings. The failure of two pools would cause the system to have 60% of the resource to deploy all strings, the failure of three pools would cause the system to have 40% of the resources to deploy all strings and the failure of four pools would cause the system to have 20% of the resources to deploy all strings. We computed the system-wide optimal resource allocations for the system offline in order to see how well the utilities achieved by the dynamic and static MMC systems compare to a system-wide optimal resource allocation.

The simulations were run such that the MMC and MCs were given sufficient time to deploy all strings after initialization. After initialization was completed, a set of pools was failed, and the resource allocation system was allowed to complete its recovery operations in response to the pool failure. We recorded the utility attained by the system immediately before the failure and after failure recovery operations completed.

We also simulated and collected data on the performance of the ARMS baseline system where gross-level resources are allocated at initialization and then no changes are made in the resource allocation. Additionally, we simulated and collected utility data for the system when the system-wide pre-computed optimal resource allocations were used.

Figure 5 contains a graph that shows 1) the ratio of the utilities achieved by the dynamic MMC allocations to the pre-computed optimal resource allocations and 2) the ratio of the utilities achieved by the static MMC allocations to the pre-computed optimal resource

allocations. As can be seen from the graph, the dynamic MMC always achieves a level of utility higher than the static MMC. Additionally, the dynamic MMC achieves a level of utility nearly as good as the globally optimal resource allocation during a wide range of operation. This implies that if the system will experience relatively low levels of resource deficiency (less than 50% resource deficiency), then our dynamic MMC methodology is nearly optimal and computationally efficient.

Figure 6 contains a graph that demonstrates how the ratio of performance for the dynamic and static MMCs varies with resource deficiency. As post-failure resource deficiency increases, the MMC system performs increasingly better with respect to the baseline system, reaching over 2x performance gains. Therefore, even though the dynamic MMC does not achieve a utility as close to the utility achieved by the global optimal resource allocation during periods of high resource deficiency, our resource allocation strategy still performs significantly better than the baseline static MMC system from the DARPA ARMS program.

While the dynamic MMC system recovers from the failures successfully and utilizes the available resources efficiently, with the increase in resource deficiency the baseline system increasingly fails to recover. On a failure in the static baseline system, the MCs attempt to reacquire lost resources to recover the utility level. If there are available resources after the failure the MCs acquire them in an uncoordinated manner. Because

larger failures affect more important strings, the baseline system is less likely to recover from these failures, whereas the viability of the coordinated MMC system becomes apparent.

6. Conclusions and Future Work

We have presented the architectural and algorithmic design of resource allocation among high-level autonomous mission controllers. We presented a heuristic approach to high-level resource management that achieves a level of utility nearly as good as the utility of an optimal pre-computed global resource allocation which is infeasible to compute on the fly.. We also compared our dynamic, heuristic approach to standard baseline approach in the ARMS system and showed that it achieves noticeably better performance under conditions of resource high deficiency.

Possible future research paths include the design of a decentralized MMC. That is, a system where the MCs negotiate gross-resource allocations without centralized coordination thereby avoiding problems associated with single point of failure in real-world deployments of DRE systems.

A further area for future research is in the area of dynamic system certification. When the dynamic resource management systems are deployed on real-world safety-critical systems, the system users need to be confident that the system is deployed and operates in accordance with system specifications [5].

7. Bibliography

- 1 P. Chandra, A. Fisher and P. Steenkiste. A Signaling Protocol for Structured Resource Allocation. In: INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. Vol. 2, pgs 522-533, 1999.
- 2 K. Dudziński and S. Walukiewicz. "Exact methods for the knapsack problem and its generalizations", European Journal of Operations Research, 28, pg 3-21, 1987
- 3 K. Krauter and R. Buyya and M. Maheswaran. A Taxonomy and Survey of Grid Resource Management Systems. *Software Practice and Experience* 32(2):135-164, 2002..
- 4 M. Maheswaran. *Quality of service driven resource management algorithms for network computing*. 1999 Int'l Conference on Parallel and Distributed Processing Technologies and Applications (PDPTA '99), June 1999, pp. 1090-- 1096.
- 5 K. Rohloff, J. Loyall, and R. Schantz. Quality Measures for Embedded Systems and Their Application to Control and Certification. In

Proceeding of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2006) Workshop on Innovative Techniques for Certification of Embedded Systems. San Jose, CA, 2006.

- 6 K. Rohloff, J. Ye, J. Loyall, and R. Schantz. A Hierarchical Control System for Dynamic Resource Management. In *Proceeding of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2006) Work in Progress Symposium*, San Jose, CA, 2006.
- 7 P. Rubel, J. Loyall, R. Schantz and M. Gillen. Fault Tolerance in a Multi-Layered DRE System: A Case Study, *Journal of Computers (JCP)*. 1(6), pg. 43-52. September 2006.
- 8 J. Stankovic, C. Lu, S. Son, and G. Tao. *The Case for Feedback Control Real-Time Scheduling*. EuroMicro Conference on Real-Time Systems, June 1999.
- 9 L. Welch, B. A. Shirazi, B. Ravindran and C. Bruggeman. DeSiDeRaTa: QoS Management Technology for Dynamic, Scalable, Dependable Real-Time Systems. In *5th IFAC Symposium on Distributed Computer Control Systems (DCCS98)*. 1998.