Deployment Automation with BLITZ

Brian Dougherty, Jules White, Jaiganesh Balasubramanian, Chris Thompson, and Douglas C. Schmidt
Department of Electrical Engineering and Computer Science,
Vanderbilt University, Nashville, TN, USA
{briand,jules,jai,schmidt}@dre.vanderbilt.edu & chris.m.thompson@vanderbilt.edu

Abstract

Minimizing the computing infrastructure (such as processors) in a distributed real-time embedded (DRE) system deployment helps reduce system size, weight, power consumption, and cost. To support software components and applications on the computing infrastructure, the hardware must provide enough processors to ensure that all applications can be scheduled without missing real-time deadlines. In addition to ensuring scheduling constraints, sufficient resources (such as memory) must be available to the software. It is hard to identify the best way(s) of deploying software components on hardware processors to minimize computing infrastructure and meet complex DRE constraints.

This paper provides the following contributions to the study of the deployment of software components to hardware in DRE systems: (1) we present an algorithmic deployment technique that minimizes the required number of processors, while adhering to real-time scheduling, resource, and co-location constraints, (2) we show how this technique can be augmented with a harmonic period heuristic to further reduce the number of required processors, and (3) we present empirical data from applying three different deployment algorithms for processor minimzation to a flight avionics DRE system.

1 Introduction

Current trends and challenges

Software engineers who develop distributed real-time and embedded (DRE) systems must carefully map software components to hardware. These software components must adhere to complex constraints, such as real-time scheduling deadlines and memory limitations, that are hard to manage when planning deployments that map the software components to hardware [1]. How software engineers choose to map software to hardware has a direct impact on the number of processors required to implement a system.

Ideally, software components for DRE systems should be deployed on as few processors as possible. Each additional processor used by a deployment adds size, weight, power consumption, and cost to the system [9]. For example, it has been estimated that each additional pound of computing infrastructure on a commercial aircraft results in a yearly loss of $100 per aircraft in fuel costs. Likewise, each pound of processor(s) requires four additional pounds of cooling, power supply, and other support hardware. Naturally, reducing fuel consumption also reduces emissions, benefiting the environment [11].

Several types of constraints must be considered when determining a valid *deployment plan*, which allocates software components to processors. First, software components deployed on each processor must not require more resources, such as memory, than the processor provides. Second, some components may have co-location constraints, requiring that one component be placed on the same processor as another component. Moreover, all components on a processor must be schedulable to assure they meet critical deadlines [10].

Existing automated deployment techniques [3, 6, 2] leveraged by software engineers do not handle all these constraints simultaneously. For example, Rate Monotonic First-Fit Scheduling [2] can guaranteee real-time scheduling constraints, but does not guarantee memory constraints or allow for forced co-location of components. Co-location of components is a critical requirement in many DRE systems. Moreover, if deploying a set of components on a processor results in CPU over-utilization, critical tasks performed by a software component may not complete by their deadline, which may be catastrophic. DRE software engineers must therefore identify deployments that meet these myriad constraints *and* minimize the total number of processors [5].

This paper provides three contributions to the study of software component deployment optimizations for DRE systems that address the challenges outlined above. First, we present the *Bin packing LocatIon Technique for processor minimiZation* (BLITZ), which uses bin packing to allocate software applications to a minimial number of processors and ensure that real-time scheduling, resource, and co-location constraints are simultaneously met. Second, we present a case study that motivates the minimization of processors in a production flight avionics DRE system. Third, we present empirical comparisons of minimizing processors for deployments with BLITZ for three different scheduling heuristics versus the simple bin-packing of one component per processor used in the avionics case study.

Paper organization. The remainder of this paper is organized as follows: Section 2 describes challenges that must be overcome to determine valid deployments; Section 3 describes the BLITZ technique we developed to minimize the number of processors in deployments; Section 4 presents empircal results from applying BLITZ to the flight avionics case study to reduce the number of processors in a deployment; and Section 5 presents concluding remarks.

2 Challenges of Component Deployment Minimization

This section summarizes the challenges of a determining a software component deployment that minimizes the number of processors in a DRE system.

Rate-monotonic scheduling constraints. To create a valid deployment, the mapping of software components to processors must guarantee that none of the software components' tasks misses its deadline. Even if rate monotonic scheduling is used, a series of components that collectively utilize less than 100% of a processor may not be schedulable. It has been shown that determining a deployment of multiple software components to multiple processors that will always meet real-time scheduling constraints is NP-Hard [3].

Task co-location constraints. In some cases, software components must be co-located on the same processor. For example, variable latency of communication between two components on separate processors may prevent real-time constraints from being honored. As a result, some components my require co-location on the same processor, which precludes the use of bin-packing algorithms that treat each software component to deploy as a separate entity.

Resource constraints. To create a validate deployment, each processor must provide the resources (such as memory) necessary for the set of software components it supports to function. Developers must ensure that components deployed to a processor do not consume more resources than are present. If each processor does not provide a sufficient amount of these resources to support all tasks on the processor, a task will not be able execute, resulting in a failure.

3 Deployment Optimization with BLITZ

The *Binpacking LocatIon Technique for processor minimiZation* (BLITZ) is a first-fit decreasing binpacking algorithm we developed to (1) assign processor utilization values that ensure schedulability if not exceeded and (2) enhance existing techniques by ensuring that multiple resource and co-location constraints are simultaneously honored.

3.1 BLITZ Bin-packing

The goal of a bin packer is to place a set of items into a minimal set of bins. Each item takes up a certain amount of space and each bin has a limited amount of space available for packing. An item can be placed in a bin as long as its placement does not exceed the remaining space in the bin. Multi-dimensional bin packing extends the algorithm by adding extra dimensions to bins and items (*e.g.*, length, width, and height) to account for additional requirements of items. For example, an item may have height corresponding to its CPU utilization and width corresponding to consumed memory.

BLITZ uses an enhanced multi-dimensional bin packing algorithm to generate valid deployments that honor multiple resource constraints and co-location constraints as well as the standard real-time scheduling constraints. In BLITZ, each processor is modeled as a bin and each independent component or co-located group of components is modeled as an item. Each bin has a dimension corresponding to the available CPU utilization. Each item has a dimension that represents the CPU utilization it requires, as well as a a dimension corresponding to each resource, such as memory, that it consumes. Each bin's size dimension corresponding to available CPU utilization is initialized 100%. The resource dimensions are set to the amount of each resource that the processor offers.

To pack the items, they are first sorted in decreasing order of utilization. Next, BLITZ attempts to place the first item in the first bin. If the placement of the item does not exceed the size of the bin (available resources and utilization) of the bin (processor), the item is placed in the bin. The dimensions of the items are then subtracted from the dimensions of the bin to reflect the addition. If the item does not fit, BLITZ attempts to insert the item into the next bin. This step is repeated until all items are packed into bins or no bin exists that can contain the item.

Burchard et al [7] describe several techniques that use component partitioning and bin-packing to reduce total required processors. This work, however, does not account for additional resource constraints, such as memory. Furthermore, these techniques do not allow for co-location constraints that require specific components to reside on the same processor.

3.2 Utilization Bounds

Conventional bin-packing algorithms assume that each bin has a static series of dimensions corresponding to available resources. For example, the amount of RAM provided by the processor is constant. Applying conventional bin-packing algorithms to software component deployment is challenge since it is hard to set a static bin dimension that guarantees the components are schedulable. Scheduling can only be modeled with a constant bin dimension of utilization if a worst-case scheduling of the system is assumed. Liu-Layland [8] have shown that a fixed bin dimension of 69.4% will guarantee schedulability but in many cases, tasks can have a higher utilization and still be schedulable.

The Liu-Layland equation states that the maximum processor utilization that guarantees scheduability is equal to $2^{1/x} - 1$, where x is the total number of components allocated to the processor. With BLITZ, each bin has a scheduling dimension that is determined by the Liu-Layland equation and the number of components currently assigned to the bin. Each item will represent at least one but possibly multiple co-located components. Each time an item is assigned to a bin, BLITZ uses the Liu-Layland formula to dynamically resize the bin's scheduling dimension according to the number of components held by the items in the bin.

If the the frequency of execution, or periodicity, of the components' execution requirements is known, higher processor utilization above the Liu-Layland bound is also possible. Components with harmonic periods (*e.g.,* periods that can be repeatedly doubled or halved to equal each other) can be allocated to the same processor with scheduability ensured, as long as the total utilization is less than or equal to 100%.

Unlike other deployment algorithms [7, 4], BLITZ uses multi-stage packing to exploit harmonic periods. In the first stage, components with harmonic periods are grouped together. In each successive stage, the components from the group with the largest aggregate processor utilization are deployed to the processors using a first-fit packing scheme. If not all periods of the components in a bin are harmonic, an item is allocated to a bin only if the utilization of its components fits within the dynamic scheduling Liu-Layland dimension and all other resource dimensions. If all component periods within a bin are harmonic, the utilization dimension is not dynamically calculated with Liu-Layland and a fixed value of 100% is used.

3.3 Co-location Constraints

To allow for component co-location constraints, BLITZ groups components that require co-location into a single item. Each item has utilization and resource consumption equal to that of the component(s) it represents. Each item remembers the components associated with it. The Liu-Layland and harmonic caculations are performed on the individual components associated with the items in a bin and not each item as a whole.

4 Empirical Results

This section presents the results of applying BLITZ to a flight avionics case study provided by Lockheed Martin Aeronautics through the SPRUCE portal (`www.sprucecommunity.org`), which provides a web-accessible tool that pairs academic researchers with industry challenge problems complete with representative project data. This case study comprised 14 processors, 89 total components, and 14 co-location constraints. We compared 2 different bin-packing strategies against both BLITZ and the baseline deployment of this avionics system, produced by the original avionics domain experts.

4.1 Experimental Platform

All algorithms were implemented in Java and all experiments were conducted on an Apple MacbookPro with a 2.4 GHz Intel Core 2 Duo processor, 2 gigabytes of RAM, running OS X version 10.5.5, and a 1.6 Java Virtual Machine (JVM) run in client mode. All experiments required less than 1 second to complete with each algorithm.

4.2 Processor Minimization with Various Scheduling Bounds

This experiment compared the following bin-packing strategies against BLITZ and the baseline deployment of the avionics system: (1) a worst-case multi-dimensional bin-packing algorithm that uses 69.4% as the utilzation bound for each bin, (2) a dynamic multi-dimensional bin-packing algorithm that uses the Liu-Leyland equation to recalculate the utilzation bound for each bin as components are added, and (3) our BLITZ technique that combines dynamic utilization bound recalculation with the harmonic period multi-stage packing.

We used each technique to generate a deployment plan for the avionics system described in Section 4. Figure 1 shows the original avionics system deployment, as well as deployment plans generated by the worst-case bin-packing algorithm, dynamic bin-packing algorithm, and BLITZ.

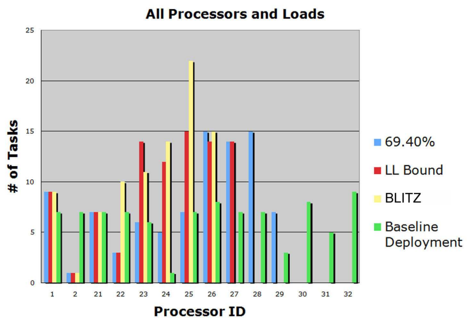

Figure 1. Deployment Plan Comparison

The BLITZ techinque required 6 less processors than the original deployment plan, 3 less processors than the worst-case bin-packing algorithm, and 1 less processor than the dynamic bin-packing algorithm.

Figure 2 shows the total reduction of processors from the original deployment plan for each algorithm. The deployment plan generated by the worst-case bin-packing algorithm reduces the required number of processors by 3 or 21.41%. The dynamic bin-packing algorithm yields a deployment plan that reduces the number of required processors by 5, or 35.71%. BLITZ reduces the number of required processors even further, generating a deployment plan that requires 6 less processors, a 43.86% reduction.

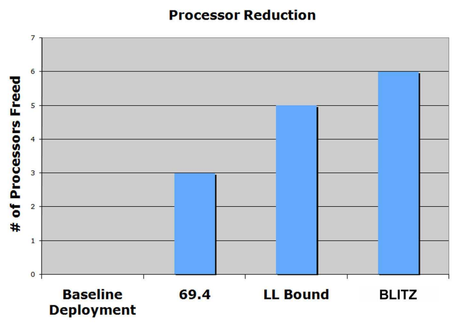

Figure 2. Scheduling Bound vs Number of Processors Reduced

5 Concluding Remarks

Determining component deployments that minimize the number of required processors is hard. This problem is exacerbated by proving that software applications are schedulable for a chosen deployment. Using bin packing algorithms, such as first-fit decreasing, the entire deployment space need not be searched. By using our BLITZ algorithm (which combines first-fit decreasing bin packing with proven utilization bounds based on data characteristics), valid and near minimal deployments can be determined.

Based on our work with BLITZ thus far, we learned the following lessons pertaining to deployment for DRE systems:

- **Grouping based on harmonic periods improves packing tightness**. BLITZ combines the Liu-Layland equation with the increased utilization bound of components with harmonic execution periods to maximize the utilization of each processor during deployment. As a result, tasks can be clustered on fewer processors, reducing the processors required.

- **Processor minimization depends on real-time benchmarks**. BLITZ has been shown to greatly reduce the required processors of a DRE system of an extensively benchmarked real-time system. Without knowledge of periodicty, resource constraints, and co-location constraints, BLITZ cannot be fully utilized. It is essential to develop tools that effectively simulate and thoroughly benchmark DRE systems before they are deployed so that the full capabilities of BLITZ can be applied.

The current version of BLITZ with example code is available in open-source form at `ascent-design-studio.googlecode.com`. The industry challenge problem that is the basis for this paper can be found at `www.sprucecommunity.org`.

References

[1] H. Beitollahi and G. Deconinck. Fault-tolerant partitioning scheduling algorithms in real-time multiprocessor systems. *Pacific Rim International Symposium on Dependable Computing, IEEE*, 0:296–304, 2006.

[2] A. Bertossi, L. Mancini, and F. Rossini. Fault-Tolerant Rate-Monotonic First-Fit Scheduling in Hard-Real-Time Systems. *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, pages 934–945, 1999.

[3] A. BURCHARD, J. LIEBEHERR, Y. OH, and S. SON. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE transactions on computers*, 44(12):1429–1442, 1995.

[4] D. De Niz and R. Rajkumar. Partitioning bin-packing algorithms for distributed real-time systems. *International Journal of Embedded Systems*, 2(3):196–208, 2006.

[5] S. Dhall and C. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, 1978.

[6] S. Lauzac, R. Melhem, and D. Mosse. Comparison of Global and Partitioning Schemes for Scheduling Rate Monotonic Tasks on a Multiprocessor. In *10th Euromicro Workshop on Real Time Systems*, pages 188–195, 1998.

[7] J. Liebeherr, A. Burchard, Y. Oh, and S. H.Son. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Trans. Comput.*, 44(12):1429–1442, 1995.

[8] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-time Environment. *JACM*, 20(1):46–61, Jan. 1973.

[9] M. Mikic-Rakic and N. Medvidovic. Architecture-Level Support for Software Component Deployment in Resource Constrained Environments. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 31–50, 2002.

[10] D. Seto, J. Lehoczky, L. Sha, and K. Shin. On task schedulability in real-time control systems. In *Real-Time Systems Symposium, 1996., 17th IEEE*, pages 13–21, 1996.

[11] N. R. C. Steering Committee for the Decadal Survey of Civil Aeronautics. *Decadal Survey of Civil Aeronautics: Foundation for the Future*. The National Academies Press, 2996.