Resource Management for Robotic Applications

Yi-Zong Ou
Dept. of CS
NTHU, Taiwan
yzou@cs.nthu.edu.tw

E. T.-H. Chu
Wen-Wei Lu
Dept. of CSIE
YunTech, Taiwan
{edwardchu, g9917722 @yuntech.edu.tw}

Jane W. S. Liu
Inst. of Info. Sc.
Academia Sinica, Taiwan
janeliu@iis.sinica.edu.tw

Ta-Chih Hung
Mech. Sys. Lab.
ITRI, Taiwan
tjhorng@itri.org.tw

Jwu-Sheng Hu
Dept. of ECE
NCTU, Taiwan
jshu@cn.nctu.edu.tw

***Abstract*—This paper presents Robotic Application Resource Management Services (RARMS), a collection of tools for integrating reusable software components of a wide class of robotic applications on Microsoft Windows, a general-purpose, commodity operating system. RARMS includes (1) a resource allocation tool called RAAPT-HV for partitioning the available processors into a specified number of virtual processors, allocating available resources to virtual processors and assigning robotic software components to virtual processors and (2) a robot-class scheduling service (RC SS) which helps independently developed components prioritize relative to each other in a way that is consistent with their timing requirements. These tools aim to provide time-sensitive components with satisfactory responsiveness in an open environment without serious impact on the performance of other components. To demonstrate the effectiveness of RARMS, we adopted for experimentation and evaluation purposes several commonly-used components of delivery robots, including face detection, speech recognition, video streaming and path planning. The results of our experiments showed that these tools can help to achieve satisfactory performance for these software components of robotic applications.**

Keywords- Real-time scheduling and resource management, Robotic software development tools, Hypervisor

I. INTRODUCTION

In the past decades, numerous service and social robotic applications have been developed. Examples include household automation robots, delivery robots, robotic pets, and robotic personal assistants. For such robots to be widely used in homes and work places, hospitals and assisted living facilities, warehouses, and so on, they must have high quality and be affordable. A proven way to keep their development cost low and time-to-market short is to build them on component-based architectures and from reusable components. Research efforts such as the ones described in [1-9] have now made this approach to building diverse service and social robots viable.

Another way to keep the cost low is to adopt an open environment on commodity platforms, such as Microsoft Windows or Linux. Indeed, these general-purpose commodity operating systems can easily integrate independently-developed components and support component-based approach to build diverse applications. Robotic applications, like many other applications with critical timing requirements, are exceptions. Effective solutions to the problems of providing bandwidth and latency guarantees to some components in an open environment without serious impact on the performance of other components remain elusive. Traditionally, one ensures the timeliness of time-critical components in a system by consistent prioritization of all computations, file accesses, messages, etc. in the system according to their real-time requirements and end-to-end tracking of their priorities across all resources (i.e., processors, disks, networks, etc.). Solutions offered by programming language standards (such as POSIX Real-Time Extensions, Real-Time Java and Ada95) and modern real-time operating systems and validation tools rely on correct and globally consistent prioritization of all application and system components sharing resources of the platform. Unfortunately, such solutions cannot always be readily applied when applications are built in an open environemnt from independently developed components, especially when it is not possible to modify the components.

This paper describes a collection of system integration tools, named Robotic Application Resource Management Services (RARMS), for scheduling component-based robotic applications and managing their resource usages in Microsoft Windows environment. RARMS aims to support the design and configuration phase of development by assisting the developer of a new robotic application to be built from independently developed components in determining whether the selected platform has sufficient resources to meet the performance requirements of the application and how CPU, memory and I/O resources should be allocated to the components to achieve the required overall performance. RARMS also aims to help developers make use of the runtime resource management mechanisms provided by the underlying platform in protecting application components running on the platform from ill effects of resource contention with other applications in the system.

Unlike many existing real-time scheduling and resource management tools (e.g.,[10,11]), tools within RARMS run at middleware and application levels and rely only on exported capabilities and application program interfaces (APIs) of the underlying platform and middleware. Consequently, RARMS is relatively easy to port across Microsoft Windows platforms. Another design objective of RARMS is that it should be particularly suited for component-based service and social robotic applications. Typical service and social robots targeted by RARMS are not severely CPU bandwidth, memory and energy limited. While being responsive to user actions and external events is essential from usability perspective, these robots do not have challengingly stringent

978-0-7695-4600-1/11 $26.00 © 2011 IEEE
DOI 10.1109/TrustCom.2011.163

and hard timing requirements. RARMS is designed to make use of these characteristics to serve the targeted robotic applications well. Hereafter, by robotic applications, we mean applications that are parts of service and social robots.

Specifically, RARMS contains two tools. They are RAAPT-HV (Resource Allocation and Application Partition Tool) and RC SS (Robotic-Class Scheduling Service). RAAPT-HV uses CPU reservation capability of Microsoft Hyper-V hypervisor [12] to provide application components with isolation and hence runtime protection against the ill effects of CPU contention to a great extent. RC SS is built on Microsoft Multi-Media Class Scheduler Service (MMC SS) [13]. Like MMC SS, RC SS uses fixed priority scheduling supported by the operating system. The more stringent is the response time requirement, the higher the priority. An advantage of RAAPT-HV is that no modification of application components is required. It can be used even when only object codes are available. The cost of isolation is the CPU and memory overheads of the hypervisor. When it is possible to make small modification of their source code, RC SS offers the applications with the easiest and most light-weight mechanism for improved real-time performance.

Following this introduction, Section II presents related works. Section III presents motivating examples and a use scenario. Sections IV and V describe the capabilities and implementations of RC SS and RAAPT-HV, respectively. Section VI summarizes the paper.

II. Related Work

As stated earlier, by making use of existing components, a component-based development process has the benefits of reduced development time and shortened time to market. This fact has motivated numerous research projects on component-based design and development of robotic applications. Examples of such project include CARMENO, CLARAty, MARIE, MIRO, ORCA, OROCOS, PLAYER, ROS and OpenRTM-aist [1-9]. Their common goal is to provide component-based architectures, development tools and platforms that can take advantages of reusable components in building diverse robotic applications. In particular, these projects typically focus on open robotic development frameworks which include messages passing between processes, connection of components and making components reusable. They provide the developers with little or no help in managing resources to ensure satisfactory performance of time-critical components.

In contrast, the RARMS focuses on resource management. RAAPT-HV aims to provide an isolated execution environment for component-based robotic applications and thus protect the robotic components from ill effects of CPU contention. It also provides tools to help developers in the design, configuration and performance tuning processes. RC SS aims to enable consistent prioritization of time-critical tasks across independently developed components. In essence, RARMS complements the above mentioned projects. The version presented in this paper is for applications running on Microsoft Windows. However, its principles can be generalized and tools revised to work in other open environments that support priority scheduling and resource partitioning.

III. Motivation Examples And Use Scenario

We present now a few representative components of delivery and telepresence robots. To keep our discussions concrete, we use the requirements of these components for motivation and illustrative purposes in the description of the use scenario(s) assumed by RAAPT-HV and RC SS, and more specifically, of how RAAPT-HV can be used during the development of applications containing these and similar components.

(A) Representative Application Components and Workloads

Without loss of generality, we consider here an advanced delivery robot for offices, hospitals and warehouses with the following capabilities: When its master issues a voice command, it turns to face the master and confirms the command. On its delivery route, it performs video surveillance functions, including detecting people it encounters and sending video of the environment back to the security office. It may also dynamically recalculate its path. Such a robot may have the following four software components. We call them *components of interest.*

- Face detection program (FDP) [14] – This program processes video frame by frame to detect faces.
- Speech recognition program (SRP) [15] – This program is used for recognition of voice commands.
- Video streaming program (VSP) [16] – This program sends video captured by an onboard camera.
- Path Planning Program (PPP) – This program runs a modified Dijkstra's shortest path algorithm when called to find new paths or modify existing path.

In addition, there are *interfering tasks* that share the platform and content for resources with components of interest. Typically, the developer wants not only to achieve satisfactory performance for each component of interest, but also needs to quantify the effects on its performance caused by interfering tasks.

The data presented in this paper were obtained by using instances of Google Chrome browsers streaming videos from YouTube to simulate interfering tasks that sometimes run at higher IRQLs (Interrupt request levels) than CPU scheduler and threads. While interrupt service routines (ISR) and deferred procedure calls (DPC) run on a CPU, all threads running on the same CPU are prevented from making progress. The invariable presence of these routines in the system is the primary reason that both CPU reservation and priority scheduling cannot provide perfect isolation and latency guarantee. We use browsers as interfering load in order to assess the degree to which components such as VSP and SRP are affected by ISRs and DPCs of applications that contain CPU tasks and spend time in network and disk I/O. The developer can choose other types of interfering tasks. For example, a busy-loop program (BLP) has no I/O. A thread running such a program simulates a CPU hog, a compute-intensive component that

can consume all the bandwidth of a CPU if allowed to do so. Through use, an extensible library of interfering tasks can be built incrementally.

Delivery robots also have components for propulsion and balancing. These components typically have hard timing constraints. We did not include these components here because they typically run on dedicated, closed platform(s), separately from components like the ones listed above.

(B) CPU Reservation Using Hyper-V

As stated earlier, RAAPT-HV is built on Microsoft Hyper-V hypervisor. It relies on Hyper-V for enforcement of resource usages at runtime. It also relies on Hyper-V for support in dividing the available logical processors into a specified number of partitions and giving each partition a percentage of reserved CPU time and specified amount of physical memory and I/O ports. Under Hyper-V, each partition has its own operating system. This is why partitions are called VM (virtual machines) in Hyper-V literature as well as in the GUI of Hyper-V. Because components running in a partition are controlled by the operating system of the partition, they are isolated to a great extent from components running outside the partition.

RAAPT-HV focuses primarily on the allocation of CPU bandwidth to partitions. A reason is that decisions on allocation of memory space and most of the required I/O ports to partitions are relatively straightforward, as it is evident from the partial view of Hyper-V GUI shown in Figure 1. (We will return shortly to explain the plot in the lower right corner of the figure.)

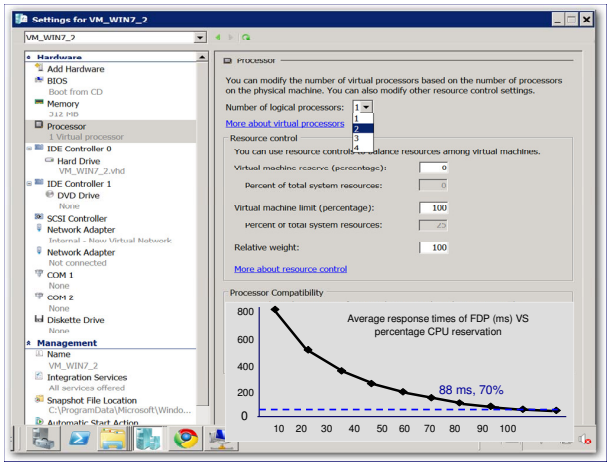

Figure 1 GUI of Hyper-V and RAAPT-HV

In contrast, the available processor parameter choices for each partition include the number of logical processors assigned to the partition. Control parameters for CPU resource include virtual machine reserve, called *CPU reservation* here. The term refers to the percentage time of the processors in a partition that is reserved for a partition. A developer can use this parameter to specify the guaranteed percentage CPU bandwidth to be provided to the components running in the partition.

The other CPU resource parameters of a partition are its *virtual machine limit* and *relative weight.* The former specifies the upper limit in percentage of processor bandwidth Hyper-V allows components in the partition to consume. The developer can improve the responsiveness of soft real-time components running on best-effort basis in a partition by making the virtual machine limit of the partition larger than its CPU reservation: With this setting, processor time not used by components running in other partitions is available to these components. The *relative weight* parameter is essentially a priority setting: A partition with a larger relative weight has a higher priority. This parameter enables the hypervisor to distribute leftover (unreserved) processor time on priority basis among partitions that have not exceeded their limits.

(C) A Use Scenario

It is evident that even with the easy to use GUI provided by Hyper-V, the task of determining the CPU reservation required by each new component on the target platform (or an existing component on a new target platform) to achieve its *acceptable performance threshold* (*APT*) is tedious and error prone. Automating these tasks as much as possible and supporting revision decisions are some of the values added by RAAPT-HV. Figure 2 shows an interactive, semi-automatic process carried out by the developer and RAAPT-HV for this determination. In the interaction diagram, time flows downward. Without loss of generality, we assume that hardware devices and I/O ports required by the *components of interest* (called *CoI* in the figure) are already installed on the target platform.

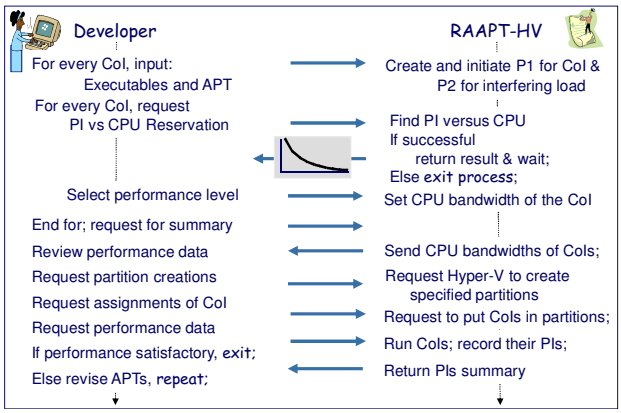

Figure 2 A use scenario of RAAPT-HV

The configuration and performance tuning process starts by the developer providing RAAPT-HV as input executable (or library functions) of each CoI, together with its APT. The *APT* is a level of component-specific performance index (PI) of each component. For instance, the PIs of FDP and SRP are average response time and success rate, respectively. To determine the CPU bandwidth required to achieve a specific level of performance for a CoI, RAAPT-HV requests Hyper-V to create two test partitions, Partition 1 (P1) for the CoI and Partition 2 (P2) for interfering load,

and set the CPU reservations of the partitions according to some procedure. An example of procedures for this purpose is `ComputeMinimumCPUReservation (CMCPUR)` to be presented in Section IV(A). A type of result generated and presented to the developer by `CMCPUR` is a plot of achievable performance level measured by the PI of the component versus required CPU reservation, exemplified by the plot for FDP shown in Figure 1. The developer can select a desired performance level by dragging a horizontal dashed line to intersect the plot at the selected level. (For the example in Figure 1, the selected performance level is 88 ms.) In response, RAAPT-HV automatically set the CPU reservation of the partition in which the component will run to the value of CPU reservation given by the plot. In this example, it is 70%.

After running the `CMCPUR` procedure for each CoI, RAAPT-HV provides a summary of selected performance levels and the corresponding required CPU bandwidths for all components of interest. Using the CPU parameter values provided by the summary and help from RAAPT-HV, the developer requests Hyper-V to create all the necessary partitions. Once all partitions are created successfully, the developer can assign components to partitions, again, by using RAAPT-HV to avoid the tedious work. After all the components of interest are in their specified partitions, the developer can use RAAPT-HV to command the components to run, record the PI of each component and organize and present the recorded data for assessment. If the PIs of all components meet their APTs, the configuration process is done. Otherwise, the PI versus CPU reservation plots generated by RAAPT-HV can help the developer adjust the desired performance levels and hence the required CPU reservations for the components used in the next iteration of configuration and performance tuning process.

IV. Robotic Class Scheduling Service

Robotic Class Scheduling Service (RC SS) is a tool that helps independently developed components prioritizes relative to each other in a way that is consistent with their timing requirements. The prototype RC SS uses MMC SS, the multi-media class scheduler service offered by Microsoft on Window Vista and later versions [17]. MMC SS enables each multimedia application to ensure that its time-critical CPU tasks are executed at appropriate priorities with respect to time-critical tasks of other multi-media applications. The rationale behind the middleware-level tool is simple: Rather than having each application requests the operating system for priority boost of its time-critical tasks, specifying the amount of boost by levels defined by the operating system, the tool accepts such requests. The input to the API `AvSetMmThreadCharacteristics()` specifies the type of the requesting application (e.g., `windows manager`, `pro audio`, `games` and `audio` in decreasing priority), not specific priority level.

In response to a request, the tool sets the priority of the caller to the priority level defined for the application type. RC SS is implemented by adding to the registry of the platform as new subkeys of `task` robotic tasks, including `SpeechRecognition`, `ObjectIdentification`, `Motion-Control`, `Localization` and `ObstacleAvoidance`. By doing so, the change does not affect multimedia applications that use MMC SS.

Figure 3 shows the performance of FDP and SRP obtained from experiments conducted to assess the effectiveness of priority scheduling and RC SS. In these experiments, we ran all four software components listed in Section II concurrently on an Intel Core 2 Quad Q8400, 2.66GHz machine with 4G RAM and Windows XP operating system. SRP and FDP were scheduled as `SpeechRecognition` and `ObjectIdentification` tasks; their threads execute at the highest real-time priorities of 31 and 30, respectively. We also ran increasing instances of chrome browsers until CPU utilization reaches 100%.

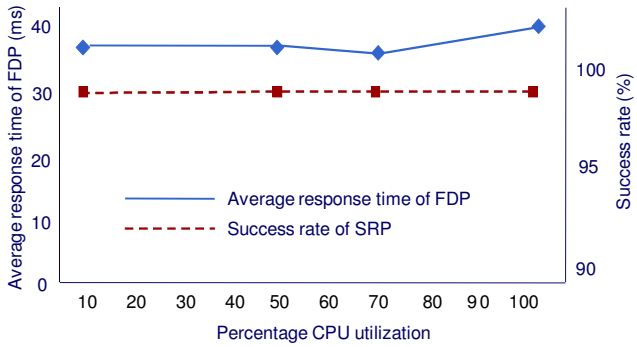

Figure 3 Effectiveness of real-time priority scheduling

It is evident from Figure 3 that by scheduling FDP and SRP components at sufficiently high real-time priorities, the effect of interfering CPU load at time-shared priorities on their performance is small. In contrast, VSP and PPP were not treated as robotic class tasks. Their performance degrades significantly when CPU utilization increases above 70% as expected since they must compete with the interfering load for CPU time.

V. Resource Allocation & Application Partition Tool

As mentioned earlier, RAAPT-HV is designed to support the iterative configuration and performance assessment and tuning process. It does so by leveraging the capabilities of Hyper-V to divide available processors into partitions and enforce resource consumption of components in each partition. It also makes use of the GUI and APIs provided by Hyper-V. Full automation of such a process by a tool is not only technically infeasible, but also not desirable. What RAAPT-HV does is to automate the parts of the process that are straightforward for a program to do, but tedious and error prone for the developer to do. Typical developer may not know how to set the combinations of CPU parameters of the partitions for the tests done in the process and what interfering tasks should be running during the tests. The tool can help to keep the process more systematic and rigorous.

(A) Functionalities for Choosing an Initial Configuration

Specifically, the tool provides the functions that support the choice of an initial percentage CPU reservation of each partition, assign components to the partitions and adjust the resources allocations of the partitions according to the needs of the components. It also helps the developer to assess the overall performance of the application, and then readjust the number and parameters of partitions to tune the overall performance.

An example is the `CMCPUR` procedure mentioned in Section II. It takes the guess work out of the step of making an initial choice of CPU reservation for the partition in which a CoI will run. For sake of simplicity, we omit the input parameters of the procedure and state it below in term of one CoI. The procedure can handle multiple components, however. Again, an APT for the CoI is given by the developer as input, as well as parameters that specify what the procedure is to do: As stated here, the procedure produces as output the required CPU reservation for the CoI to meet the APT or a declaration that the platform cannot meet the requirement. When used for this purpose, the interfering threads execute a busy-loop program (BLP) in partition 2 (P2).

`ComputeMinimumCPUReservation(CMCPUR):`

1) Request Hyper-V to create two partitions; each partition has as many processors as there are logical processors in the platform. Assign the CoI to run in partition 1 (P1).
2) Start an interfering thread running on each processor in partition 2 (P2) so that the interfering threads in P2 can use all the CPU time on each processor.
3) For the given APT of the component
   i. Set the `Current_Reservation (CR)` of P1 to the minimum reservation (say 10%) and the reservation of P2 to 100 – `CR`.
   ii. Run the CoI in P1 and measure and record the performance index (PI) (e.g., the maximum, average and minimum response time for the FDP) of the component.
   iii. If the measured PI is poorer than APT
      - If the `CR` of P1 is less than 100 %, increase the `CR` by a specified increment (say 10%); set CPU reservation of P2 to 100 – `CR`; go back to Step 3)ii;
      - Else declare "Threshold performance is not achievable on this platform."; exit.

      Else, set the CPU reservation of PI to `CR` and update configuration parameters of the CoI; exit.

The `CMCPUR` also can be invoked to run through a specified or full range of CPU reservation (e.g., 10-100%) for the given CoI, rather than stopping when the given APT is reached. Thus used, `CMCPUR` provides the performance measurement capability of RAAPT-HV. The data on the PI of the CoI as a function of CPU reservation recorded during each run is exported as a data log file. RAAPT-HV generates a plot exemplified by the one shown in Figure 1 from the log file. As stated in Section II, the plot of PI versus percentage CPU reservation enables the developer to make informed decision during performance tuning phase on how to adjust the APT of the CoI to tradeoff performance for CPU bandwidth to be provided to the CoI.

The data log file generated during runs of `CMCPUR` procedure is integrated with runtime environment setup file to produce a configuration log file for each CoI. For a component with source code, the developer can modify the source so that the files are integrated automatically. For components without source code, RAAPT-HV provides the developer with both the environment setup information and rules/guidelines for manual integration.

(B) Functionalities for Performance Tuning

Performance tuning is a mandatory step of a design and configuration process for many reasons, including making the end product as good as possible or as inexpensive as possible. Performance tuning starts after the developer ran the `CMCPUR` procedure for each CoI and a percentage CPU reservation of the partition for the CoI has been selected. If the total CPU reservation of all partitions is less than 100%, the developer may choose to take advantage of the extra CPU bandwidth to improve the APTs of some CoIs, and thus improve the overall quality of the robotic application. Again, the PI versus CPU reservation plot of each CoI and summary report generated by RAAPT-HV from the configuration log files provides the information needed to support tradeoff decisions.

If the total CPU reservation is larger than 100% and adding more hardware resource to increase available CPU bandwidth is not an option, the developer has no choice but to decrease the APTs of some CoIs and/or have some CoIs share partition(s). The procedure described partially below is an option RAAPT-HV provides to support such decisions. The developer may want to have it run when that total CPU reservation of all partitions in the initial configuration exceeds 100%. (The procedure is called from the RAAPT-HV GUI, which prompts the developer for decision on whether to run.) The input of the procedure includes the APT and performance criticality of each CoI. By combining two CoIs, we mean that they are to be treated by the tool as a unit, (i.e., a *compound CoI*) to be assigned together to a partition. By having the procedure invoked repeatedly with different CoIs as input, the developer can have more than two CoIs combined.

`TunePerformanceToUseLessResource(TPTULR):`

1) If the total CPU reservation of CoIs is less than 100%, exit;
   Else, do `CombineComponents;`
      If it is successful, declare successful; exit;
      Else, declare "Unable to reduce total CPU reservation by combining CoIs"; exit ;
2) `CombineComponents (CC):`
   i. Create two partitions P1 and P2 and assign all logical

processors to each partition;

ii. Sort all CoIs in non-increasing order according to their performance criticalities. Let CoI-L with CPU reservation Reserve-L be the last CoI on the list, and CoI-NL with CPU reservation Reserve-NL be the next to the last one.

iii. Start as many interfering threads running as there are processors in partition 2 (P2); assign CoI-L to P1;

iv. Assign CoI-NL to P1; set `CR` of P1 to (Reserve-L + Reserve-NL) and the reservation of P2 to (100% - `CR`).

v. Run CoI-L and CoI-NL; measure and record their performance indexes PI-L and PI-NL;

(a) If both PIs are worse than the corresponding APTs,
- Do `TryAgain`;

(b) Else,
- Invoke `CMCPUR` to get the CPU reservation required to meet the APTs of both components;
- If `CMCPUR` succeeds, return successful;
- Else do `TryAgain`;

3) `TryAgain macro`: Remove CoI-NL the sorted CoI list; if the list contains only CoI-L, return unsuccessful; else name the next to the last CoI on the list CoI-NL; go back to 2)iv;.

The rationale behind `CC` is that the runtime overhead of switching between the partitions on the Hyper-V can be reduced by having less performance critical components (e.g., PPP and VSP described earlier) shared a partition, and thus reducing the number of partitions. In order to be more user-friendly, RAAPT-HV GUI presents the results of the `TPTULR` in forms of plots, tables and spreadsheets and enables the developer to input selections and changes in both interactive and batch modes.

(C) Experimental Evaluation

Hyper-V was designed primarily for server applications. Over intervals much longer than typical ISRs and DPCs, it can provide CPU bandwidth guarantee to each partition. However, it cannot provide perfect isolation to time-critical components of robotic applications because it is not designed to guarantee on-time delivery of the CPU time within typical relative deadlines (e.g., 10 to 200 ms) of these components. This is the reason that an essential step of performance assessment and tuning process is to determine whether the effects of interfering loads containing ISRs and DPCs on the performance of CoIs are tolerable. When run in the performance measurement mode, `CMCPUR` procedure supports this step in part. For this purpose, the developer also needs representative interfering loads that may run concurrently with components of interests outside of their partitions.

One kind of interfering load consists of threads each of which runs a BLP program. Busy-loop programs simulate compute-intensive components that have little or with no I/O. Hyper-V can enforce their CPU consumption precisely. RAAPT-HV uses this type of interfering load when `CMCPUR` is called to estimate the required CPU reservation for a CoI. For performance assessment purposes, plots generated by `CMCPUR` using this type of interfering loads provide the developer with a basis for quantifying the ill effects of other types of interfering loads.

In experiments for the purpose of evaluating this limitation of Hyper-V and hence RAAPT-HV, we used as interfering loads instances of Google Chrome browser streaming videos from YouTube. In these experiments, the system was divided into two partitions, each of which uses all the cores. We ran the CoI(s) (e.g., one or more software components in Section II) in partition 1 and an interfering load in partition 2. Table 1 summarizes the configurations of the experiments done with FDP (face detection program) as CoI. In experiment 1, we ran `CMCPUR` with BLP as interfering load. The plot of average response time versus percentage CPU reservation of P1 thus obtained is the one shown in right lower corner of Figure 1.

Common configurations of Partition 1 and 2 in Experiment 1 and 2				
Hardware platform	Intel Core 2 Duo supports Intel VT with 4 GB RAM, 500GB, 7200 rpm Hard Disk			
Parent partition	Microsoft Server 2008R2 with Hyper-V		Child partitions	Windows XP SP3
Configuration of each partition	logical processors: 2, memory: 512MB, CPU Limit:100, Relative weight: 100			
Different configurations of Partition 1 and 2 in Experiment 1 and 2				
	Experiment1		Experiment2	
	Partition1	Partition2	Partition1	Partition2
CPU Reserve	Variable %	100 - Variable%	Variable%	100 -Variable%
Programs on the Partition	FDP	2 BLPs	FDP	Chrome Browsers

Table 1 Configurations of Experiment 1 and 2

In experiment 2, we set the CPU reservations of the partitions to several pairs of values. For each pair, we ran the FDP in partition 1 by itself, and 0, 1, 2, 5 and 10 instances of Chrome browsers in partition 2. Figure 4 shows the average response time of FDP as a function of the number of browser instances. It is evident that ISRs and DPCs of the browsers prevent the Hyper-V from having complete control over delivery of CPU time to partition 1. We note that the performance of FDP actually improves with the number of running browsers. A reason is that the blocking time of browsers increases with the number of browser instances, leaving more CPU time of partition 2 unused and the CPU reserve parameters of partition 1 were set to allow the partition to use the unused time.

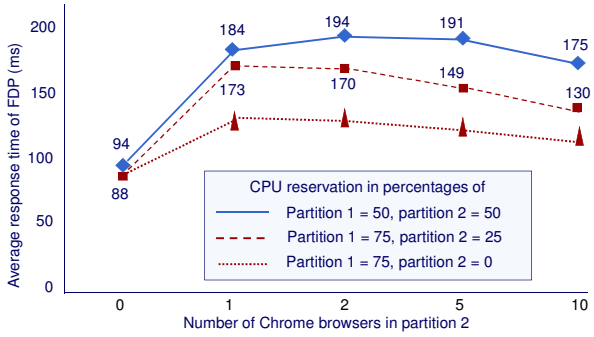

Figure 4 Effects of interfering load

VI. SUMMARY AND FUTURE PLANS

We described in this paper the capabilities of RAAPT-HV and RC SS, the component tools of RARMS, for managing resources usages of performance-critical components of robotic systems. RAAPT-HV supports a semi-automatic iterative configuration and performance assessment and tuning process. When it is not possible to modify source code of application components, it offers a way to reduce the ill-effects of resource contention. When it is possible to make small source code modification, RC SS offers the applications with the easiest and most light-weight mechanism for improved real-time performance.

In the future, we plan to fully exploit all the CPU reserve parameters, rather than relying solely on CPU reservation. Our goal is to make RAAPT-HV as effective as possible in enforcement of CPU usages of all partitions. The RAAPT-HV user interface is built on the GUI provided by Hyper-V. We need to make it user friendly and effective in its interaction with the developer in each step of the configuration and performance tuning process. The current version of RC SS does not fully exploit multiple cores within a platform. It allows the operating system to choose where each component runs. We plan to experiment with pinning down components on specified CPUs and provide the user interface needed to automate as much as possible this choice in addition to choices in priorities.

VII. ACKNOLEDGEMENT

The authors would like to thank the anonymous reviewers for their constructive suggestions. This work was supported in part by Industrial Technology Research Institute and National Science Council, Taiwan (ROC), under Grant ITRI 100A0070SB, NSC 100-2218-E-224-001-MY2 and NSC 100-2219-E-224-001.

REFERENCES

[1] Montemerlo, M., N. Roy and S. Thrun, "Perspectives on standardization in mobile robot programming: the CARMEN toolkit," in *Proc. IORS*, 2003.

[2] Nesnas, I. A. D. *et al.*, "CLARAty and challenges in developing interoperable robotic software," in *Proc. IORS*, 2003.

[3] Cote, C., *et al*., "Robotic software integration using MARIE," *International Journal of Advanced Robotic Systems*, vol. 3, 2006.

[4] Utz, H., S. Sablatnog, S. Enderle, and G. Kraetzschmar, "MIRO – middleware for mobile robot applications," *IEEE Trans. on Robotics and Automation*, vol. 18, p493~ 497, 2002.

[5] Makarenko, A., A. Brooks, and T. Kaupp, "ORCA: components for robotics," In *Proc. IORS, 2006.*

[6] Bruyninckx, H.; , "Open robot control software: the OROCOS project," *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on* , vol.3, no., pp. 2523- 2528 vol.3, 2001

[7] Vaughan, R.T.; Gerkey, B.P.; Howard, A.; , "On device abstractions for portable, reusable robot code," *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on* , vol.3, no., pp. 2421- 2427 vol.3, 27-31 Oct. 2003

[8] Cousins, S.; Gerkey, B.; Conley, K.; Garage, W.; , "Sharing Software with ROS [ROS Topics]," *Robotics & Automation Magazine, IEEE* , vol.17, no.2, pp.12-14, June 2010

[9] Ando, N.; Suehiro, T.; Kitagaki, K.; Kotoku, T.; Woo-Keun Yoon; , "RT-Component Object Model in RT-Middleware—Distributed Component Middleware for RT (Robot Technology)," *Computational Intelligence in Robotics and Automation, 2005. CIRA 2005. Proceedings. 2005 IEEE International Symposium on* , vol., no., pp.457-462, 30-30 June 2005

[10] Lakshmanan, K. and R. Rajkumar, "Distributed Resource Kernels: OS Support for End-To-End Resource Isolation", In *Proc. RTAS, 2008.*

[11] Deng Z., Jane W.-S. Liu, Lynn Y. Zhang, Mouna Seri and Alban Frei, "An Open Environment for Real-Time Applications", *Journal of Real-Time Systems*, vol. 16, no. 2-3, p.155~185,1999.

[12] Microsoft Hyper-V, http://www.microsoft.com/hyper-v-server/en/us/default.aspx

[13] Darryl E Havens, "Class scheduler for increasing the probability of processor access by time-sensitive processes", *USA Patent No. 7802256*, 2010

[14] OpenCV face detection, http://opencv.willowgarage.com/wiki/FaceDetection

[15] Speech recognition, http://www.codeproject.com/KB/audio-video/tambiSR.aspx

[16] Microsoft Windows Media Encoder 9, http://www.microsoft.com/expression/products/EncoderPro_Overview.aspx

[17] Microsoft, MMC SS (Media Class Scheduler Service), http://msdn.microsoft.com/en-us/library/ms684247(VS.85).aspx