Dynamic Task Allocation in Clusters of Embedded Smart Cameras

M. Bramberger, B. Rinner
{bramberger, rinner}@iti.tugraz.at
Institute for Technical Informatics
Graz University of Technology
Graz, AUSTRIA

H. Schwabach
helmut.schwabach@arcs.ac.at
ARC seibersdorf research
Seibersdorf, AUSTRIA

Abstract – This paper presents a dynamic task allocation method for smart cameras targeting traffic surveillance. Since our target platforms are distributed embedded systems with limited resources, the task allocation has to be light-weight, flexible as well as scalable and has to support real-time requirements. Therefore, surveillance tasks are not allocated to smart cameras directly, but to groups of smart cameras, so called surveillance clusters. We formulate the allocation problem as a distributed constraint satisfaction problem (DCSP) and present a distributed method for finding feasible allocations. Finally, a cost function is used to determine the optimal allocation of tasks. We have realized this dynamic task allocation using heterogeneous, mobile agents which utilize their agencies and our embedded software framework to find the most appropriate mapping of tasks in a distributed manner. The dynamic task allocation has been implemented on our smart cameras (SmartCam) which are comprised of a network processor and several digital signal processors (DSPs) and provide a complex software framework.

Kurzfassung – Diese Arbeit präsentiert eine Methodik zur dynamischen Verteilung von Aufgaben in einem Verkehrs-überwachungssystem, welches aus intelligenten Kameras besteht. Da wir uns hier mit verteilten, eingebetteten Systemen mit eingeschränkten Ressourcen beschäftigen, muß diese Aufgabenverteilung schlank, flexibel und skalierbar sein. Das System muß außerdem Echtzeitanforderungen genügen, sowohl die Verteilung der Aufgaben, wie die Aufgaben selbst. Um eine effektive Verteilung von Aufgaben gewährleisten zu können, werden Aufgaben nicht einer Kamera dirket zugeordnet, sondern einer Gruppe von intelligenten Kamera, einem sogenannten Überwachungs-Cluster (*Surveillance Cluster*). Wir zeigen ausserdem, dass die Generalisierung dieses Problems ein verteiltes *constraint-satisfaction-problem (CSP)* ist, welches wir mittels einer verteilten Methodik lösen, um *alle* plausiblen Verteilungen von Aufgaben zu untersuchen. Zur Bewertung der einzelnen Verteilungen verwenden wir eine Kostenfunktion um die optimale Lösung des Problems zu finden.
Realisiert wurde das System mittels heterogenen, mobilen Agenten, welche unser Software-Framework nutzen, um die optimale Zuordnung der Aufgaben in verteilter Art und Weise zu finden. Dieses System haben wir auf Prototypen unserer intelligenten Kamera (SmartCam) implementiert, welche aus einem Netzwerkprozessor und einer variablen Anzahl von digitalen Signalprozessoren besteht.

Keywords: Real-time mapping; agent system; embedded system; real-time.

I. INTRODUCTION

Surveillance systems currently undergo a dramatic shift. Traditional surveillance systems of the first and second generation have employed analog CCTV cameras, which transferred the analog video data to digital base stations where the video analysis, storage and retrieval takes place. Current semiconductor technology enables surveillance systems to leap forward to third generation systems which employ digital cameras with on-board video compression and communication capabilities. Smart cameras [9] [2] even go one step further; they not only capture and compress the grabbed video stream, but also perform sophisticated, real-time, on-board video analysis of the captured scene. Smart cameras help in (i) reducing the communication bandwidth between camera and base station, (ii) decentralizing the overall surveillance system and hence to improve the fault tolerance, as well as (iii) realizing more surveillance tasks than with traditional cameras.

Typical tasks to be run in a surveillance system targeting traffic surveillance include MPEG-4 video compression, various video analysis algorithms such as accident detection, wrong-way drivers detection and stationary vehicle detection. Additionally, traffic parameters such as average speed, lane occupancy and vehicle classification are often required. Since computing power is limited we may not allocate all tasks to the cameras.

This paper presents a resource-aware dynamic task allocation system for smart cameras targeting traffic surveillance. Since our target platforms are distributed embedded systems with limited resources, the task allocation has to be light-weight, flexible and scalable as well as has to support real-time requirements. Therefore, surveillance tasks are not allocated to smart cameras directly, but to groups of smart cameras, so called surveillance clusters. We formulate the allocation problem as a distributed constraint satisfaction problem (DCSP) and present a distributed method for finding feasible allocations. Finally, a cost function is used to determine the optimal allocation of tasks. We have realized this dynamic task allocation using heterogeneous, mobile agents which utilize their agencies and our embedded software framework to find the most appropriate mapping of tasks in a distributed manner The main contributions of this ongoing research project include (i) the distributed agent-based approach for determining all feasible allocations of a DCSP, (ii) the evaluation of a complex cost function considering the limited resources of the embedded platform in detail, (iii) the integration of a mobile agent system in our embedded software framework. The dynamic task allocation has been implemented on our smart cameras (SmartCam [3]) which are comprised of a network

processor and several digital signal processors (DSPs) and provide a complex software framework. The remainder of this paper is organized as follows: Section 1.1 sketches related work. Section 2 briefly presents hardware and software of our SmartCam. Section 3 discusses the advantages of grouping smart cameras to surveillance clusters. Section 4 describes the DCSP approach and focuses on finding feasible allocations of tasks and the cost function. Section 5 and 6 present the implementation and the experimental results, respectively. Section 7 concludes the paper with a summary and an outlook on future work.

A. Related Work

Agents-based load distribution has been an active research in the last decade. An overview of agent standards and available platforms is presented in [6]. WYA (While You're Away) [8] is based on the NOMADS mobile agent system. It provides dynamic load balancing by mobile agents, which utilize a central coordinator to move between workstations. Qin et al [7] identified the amount I/O-traffic as an important issue for dynamic load balancing beside CPU-load and memory. Chow and Kwok [4] introduced the affinity of an agent to its machine and used a credit-based scheme to determine the agents to be migrated by using a central host station. Agents used in surveillance systems were proposed by the MODEST consortium [1], where mobile agents were used to track vehicles using PC based platforms. Distributed constraint satisfaction problems have been analyzed by Yokoo et al. in [10] for variablesplit CSPs. The presented asynchronous backtracking algorithm used autonomous agents, however, the high communication effort between the agents is impractical for real-time systems.

II. THE SMART CAMERA

Smart cameras are the core components of a 3rd generation surveillance system. These cameras perform video sensing, high-level video analysis and compression and transfer the compressed data as well as the results of the video analysis to a central monitoring station. The video analysis tasks implemented in the cameras clearly depend on the overall surveillance application and may include accident detection, vehicle tracking and the computation of traffic statistics. Most of these tasks, however, require a very high computing performance on the cameras.

A. Hardware Architecture

Our smart camera has been designed as a low-power, high-performance embedded system. As depicted in Figure 1, the smart camera consists of three main units. (1) The sensing unit, (2) the processing unit, and (3) the communication unit. A high-dynamic, monochrome CMOS image sensor is the heart of the sensing unit. It delivers images with VGA resolution at 25 frames per second via a FIFO memory to the processing unit. Real-time video analysis and compression is performed at the processing unit which is equipped with up to four digital signal processors (DSPs) TMS320C6415 from Texas Instruments. The DSPs deliver an aggregate computing performance of almost 20 GIPS while keeping the power

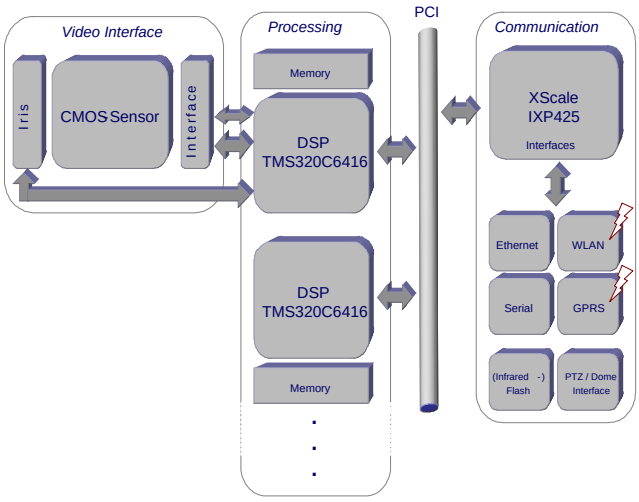

Figure 1: The architecture of the smart camera

consumption low. The DSPs are coupled via a local PCI bus which serves also as connection to the network processor (Intel XScale IXP425) in the communication unit. The communication unit provides access to the camera's environment. The communication of the smart camera is basically two-fold. First, the communication unit manages the internal communication between either the DSPs and the DSPs and the network processor. Second, it manages the external communication, which is usually IP-based. The XScale processor is operated by Linux due to large number of available tools and applications available under this operating system. [3] provides a more detailed insight into the hard- and software architecture of the smart camera.

B. Software Architecture

The software architecture of our smart camera is designed for flexibility and reconfigurability. The software architecture consists of several layers which can be grouped into: (1) The DSP framework, which is implemented on the DSPs, and (2) the SmartCam framework, running on the network processor. DSP Framework The main purpose of the DSP is (i) the abstraction of hardware and communication channels, (ii) the support for dynamic loading and unloading of applications, and (iii) the management of on-chip and off-chip resources of the DSP system by utilizing Texas Instruments Reference Framework 5 [5]. SmartCam Framework The SmartCam framework (cp. figure 3) serves the following purpose: First, it provides abstraction of the DSPs to ensure platform independence of the agent-system and application layers. Second, the application layer uses the provided communication methods (messaging to the DSPs and IP-based communication to outer world) to exchange information, or work as a relay service, respectively. Finally, the agent-system layer is run on top of Java, whereas the agents are run as a part of the agent platform.

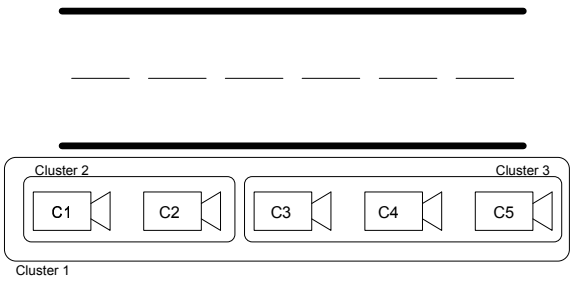

Figure 2: A surveillance scene with tree clusters

III. SURVEILLANCE SYSTEM ARCHITECTURE

The architecture of the surveillance system consists of a large number of smart cameras deployed alongside highways or in tunnels. Since these smart cameras have limited computational resources, it is not possible to run all required surveillance tasks on a smart camera. Therefore, physically co-located smart cameras are combined into logical groups, so called surveillance clusters. Consequently, sets of surveillance tasks (e.g. accident detection, vehicle counting, vehicle classification) are then allocated to surveillance clusters. This is feasible, since events, observed by co-located cameras are causally associated with each other. Therefore, it is not important on which smart camera a surveillance task is allocated, as long as these surveillance clusters do not span a too large area. Not all surveillance tasks require small surveillance clusters; classifying and counting of vehicles, for example, may be spread over a larger area, while accident or fire detection tasks are usually allocated to smaller clusters. Therefore, a smart camera may be a member of several surveillance clusters (cp. Figure 2). The allocation of surveillance tasks to smart cameras is done dynamically during runtime by the presented task allocation system (see section 4), which is distributed over all smart cameras.

In contrast to the presented surveillance tasks, which may be allocated to any smart camera in the surveillance cluster, there are also several surveillance tasks which are required to run on a specific smart camera. These tasks possess an affinity to a scene or a camera, respectively. Tracking algorithms, for example, require to be run on a specific camera, which observes the tracked object. These scene-affine tasks are also contained in a surveillance cluster, however, the task allocation system should allocate these tasks to the required smart cameras. Note that scene-affine tasks might be allocated to a different camera. In this case, however, the communication load would significantly be increased because raw images would have to be transferred from one camera to another.

IV. TASK ALLOCATION

The goal of the task allocation system is to find a mapping of tasks to smart cameras which satisfies all requirements and is optimal with respect to a specified metric, i.e., some cost function. Since the surveillance tasks have firm real-time requirements, the task allocation system has to take care that no deadlines are missed due to an overload of a camera or a camera's subsystem, respectively. The re-allocation of tasks may be necessary due to events, raised by hardware or software: (1) Hardware events usually originate from changed resource availability due to added or removed cameras, hardware breakdown, or re-usability of recovered resources. (2) Software events are caused by changes in resource requirements due to changes in the task set of the surveillance cluster, or because of changes in the quality-of-service level (QoS) of tasks, i.e., due to detected events in the observed scene. The allocation of tasks to smart cameras is done in two steps. (1) In the first step, all feasible allocations of tasks to smart cameras (allocations where all real-time requirements are satisfied) are determined. (2) In the second step, the optimal allocation of tasks is chosen by using a cost function.

A. Find Feasible Allocations

The determination of feasible allocation of tasks to smart cameras is a distributed constraint satisfaction problem (CSP) [10]. CSPs are defined by a set of variables $T=\{T_1,\ldots,T_n\}$, which hold values of a finite domain D, and a set of k constraints $C=\{C_1,\ldots,C_k\}$. In our case, the surveillance tasks to allocated to the cameras of the surveillance cluster are the variables T, and the stored value is the identifier of the hosting smart camera. The domain is defined as $D=\{1,\ldots,m\}$, therefore, $T_i=j$ indicates the allocation of task i to smart camera j. A feasible allocation of tasks to cameras means, that all resource requirements can be satisfied, since an overload of even a single resource would lead to a violation of real-time deadlines. In order to formulte the constraints, two functions which determine the resource requirements of the tasks and the resource availability on the smart cameras are defined. Therefore, *req(Res, i)* determines the requirements to resource *Res* (where $\mathrm{Res} \in \{CPU, Mem, DMA, IRQ\}$)of task *i*, *while* the availability of resource *Res* on smart camera *j* is determined by *avail(Res, j)*. The feasible allocations of tasks are all combinations of tasks, where (i) all resource requirements of tasks are met, while (2) no resource on any smart camera is overloaded, and (3) all tasks are allocated to a smart camera.

In order to distribute the determination of feasible allocations to all smart cameras, the domain D is split into m sub-domains $D_1,\ldots,D_m$, each comprised of a single value $D_l=l$. This way, each smart camera determines all feasible allocations of tasks on itself, which is done in parallel on the smart cameras. Finally, all feasible allocations are merged into a set of feasible allocations, which (1) include all required tasks, and (2) do not allocate a task to two smar cameras concurrently.

B. Find Optimal Allocation

From the set of feasible allocations the optimal allocation concerning (i) the number of migrations, (ii) the amount of communication between the cameras, and (iii) the quality-of-service has to be found. Therefore, we use a cost function, which takes (1) resource cost, (2) data-transfer cost, (3) migration cost, (4) affinity cost, and (5) quality-of-service cost into account. Consequently, we calculate the oveall cost for each feasible allocation and

choose the one with the lowest cost as new allocation of tasks. Note that the weight of each cost can be scaled, therefore, it is possible to set an emphasis on a certain cost during optimization.

C. Reallocation Scenarios

As mentioned before, two scenarios are possible, where a reallocation of tasks is necessary. However, we handle the scenarios differently to improve the performance of the system.

Increased Requirements / Decreased Resource Availability

Increased requirements or decreased resource availability indicates a higher system load. Therefore, the number of feasible allocations will decrease, since less combinations of tasks will satisfy the constraints of the CSP. Therefore, the value of the changed resource or availability is updated in the set of feasible allocations F. Finally, the feasible allocations are re-checked and infeasible allocations are removed from F.

This re-checking process does not require to recalculate all possible allocations, therefore, the time required for determination of the new allocation is reduced dramatically.

Decreased Requirements / Increased Resource Availability

In contrast to the upper scenario, the decrease of requirements or increase of resource availability can be seen as a reduced load of the system. Consequently, the number of both, partial and complete, allocations will increase. Therefore, all feasible allocations have to be re-computed. This approach leads to longer execution times, however, as the service quality will rise by this reallocation, the real-time deadlines are not as firm as in the upper scenario.

V. IMPLEMENTATION

For the implementation of our surveillance system we have chosen to use mobile agent technology. Mobile agents are most suitable for our system, since they support mobility, autonomy, and platform independence.

A. Agent System

We have chosen to use the diet-agents (see http://diet-agents.sf.net) system since it includes all required features like mobility, autonomy, and platform independence and is reasonable small, which is inevitable for embedded systems. However, extensions to the agent system were necessary to add the support for the DSPs, and the decentralized management of the surveillance clusters.

DSP Agencies

Agencies provide the environment for agents to live in. An extension to standard agencies is required to support multiple surveillance clusters. Therefore, a logical grouping of agents within the agency (cp. surveillance clusters A, B and x in Figure 3) is desirable. We have implemented a cluster-information agent (CIA), which hosts local knowledge of other cameras belonging to the surveillance cluster, and provides lookup services for agents within the surveillance cluster.

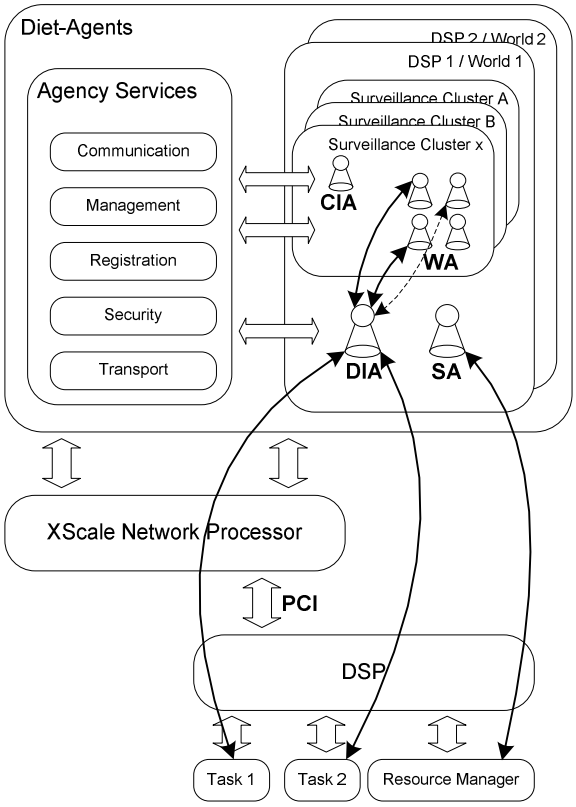

Figure 3: The DSP Agency

The integration of the DSPs into the agent system is done by one DSP integration agent (DIA) per DSP in the system. This DIA is a stationary agent, which manages message registrations, message dispatching, and binary downloads to the DSP.

Two basic agent types are deployed within the agent system: (1) Management-oriented SmartCamagents, which are not included in the task allocation system, and (2) DSP-agents, which implement algorithmic functionality to be executed on a DSP. Since the implementation of the SmartCam agents is quite straightforward, only the DSP agents will be discussed in detail in the next section.

DSP Agents

DSP agents are an enhancement of usual agents, as they are the base of the task allocation system. Therefore, DSP agents include their requirements to the system and their cost parameters. Additionally, the cost calculation functionality is added to these agents(cp. Figure 4). As the computational intensive parts of these agents are executed on the DSPs, the agents contain a binary, which is downloaded to the DSP and dynamically integrated into the system by the DSP framework. Therefore, the agents maintains communication channels to the DSP by using the world's DSP integration agent (DIA). In case of migration the agent notifies the DSPs task to stop computation and to transmit the persistent intermediate results to the agent. After migration the agent loads the binary to the DSP and transmits the stored intermediate results to the DSP. This

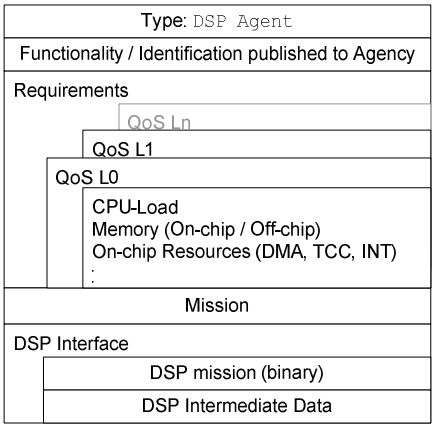

Figure 4: A DSP Agent

enables the algorithm to continue computation or to use the intermediate results as new starting values, respectively.

B. Task Allocation

This section outlines the actions taken during a reconfiguration of a surveillance cluster. Since our goal is a flexible, scalable and distributed implementation we have chosen to realize the task allocation system using mobile agents.

A new allocation is always initiated by a smart camera, either due to a hardware event (changes of availability) or a software event (changes of requirements). This camera initiates the reallocation by broadcasting the request-for-requirements to all agents, that encapsulate the surveillance tasks, within the surveillance cluster. After the agents receive this request, they broadcast their requirements to all smart cameras in their surveillance cluster. Additionally, all agents calculate their costs for every smart camera and transmit the cost values to the initiating camera. The smart cameras determine in parallel the partial allocations, which have to be merged in the next step. As the smart cameras are comprised of more DSPs, the partial allocations for the DSPs are merged. To further merge the partial allocations, half of the smart cameras (determined during the creation of the surveillance cluster) create allocationmerging agents (AMA), which pick up the merged partial allocation and migrate to predetermined smart cameras (which comprise the other half of the surveillance cluster), where they grab the partial allocation, and merge both, the local and the included partial allocations. After the merging process, the agents migrate to the next smart camera, as defined by the fixed itinerary, and merge their partial allocation with the partial allocation provided by another AMA. This way the partial allocations are merged to a final allocation, whereas the last allocation merge is done on the initiating smart camera. The merging process corresponds to a binary tree, therefore ld(m) (where m is the number of smart cameras) sequential steps have to be done.

The initiating smart camera has to determine the most appropriate task allocation, therefore, the costs, as submitted by the agents, are accumulated for every task allocation. Consequently, the task allocation with the lowest cost is selected as the new task allocation. Since the selection of the most appropriate task allocation is based on the costs of the agents, the desired optimization goal like the number of migrations, degradation of quality-of-service, or balanced use of resources can be achieved by adapting the scaling factors of the cost classes. Finally, the new task allocation is broadcast to all smart cameras and agents, which may update their QoS level, or migrate to another smart camera then. To enable the fast reconfiguration by removing feasible task allocations, also the set of feasible task allocations is broadcast to all smart cameras in the surveillance cluster after the system has settled.

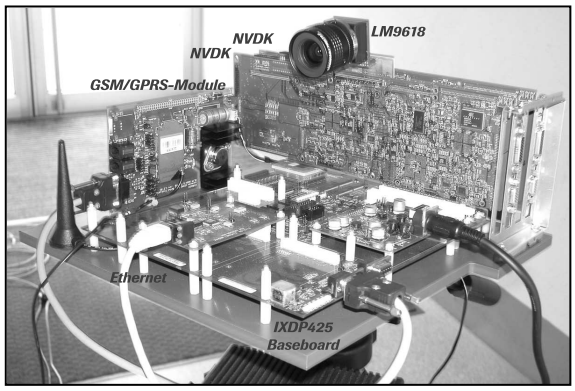

Figure 5: The prototype of the smart camera

C. Harware Setup

To verify, test and evaluate the presented agent system, we have used two hardware platforms. A prototype of our smart camera and PCs equipped with DSP boards. The prototype of our smart camera consists of an Intel IXDP425 Development Board, which is equipped with an Intel IXP425 network processor running at 533 MHz. This processor features onchip Ethernet MACs, serial communication ports and a PCI host controller. The board is operated with Linux Kernel 2.6.8.1, which allows the usage of standard software packages, and enables interoperability with PC-based Linux systems. To enable the interaction with the DSPs, the SmartCam framework is run on top of Linux. The board supports up to four DSP boards, however, our prototype is equipped with two Network Video Development Kits (NVDK) from ATEME. Each board is comprised of a TMS320C6416 DSP from Texas Instruments, running at 600 MHz, with a total of 264 MB of on-board memory. Image acquisition is done using the National Semiconductor LM9618 monochrome CMOS image sensor, which is connected to one of the DSP boards.

Due to the lack of additional smart camera prototypes, we are using two Pentium-III personal computers (PCs) running at 1 GHz, which are equipped with one Network Video Development Kit (NVDK) from ATEME. We are using the analog video inputs from the DSP card to grab the required video data. These video inputs are fed by analog cameras or VCRs.

VI. EXPERIMENTS

In order to verify and evaluate the presented task allocation system, we created a surveillance cluster, comprised of the smart camera prototype and two PCs. We have used four different agent types for our experiments: (1) A MPEG-4 video compression agent, (2) a stationary-vehicle detection (StVD) agent, (3) a vehicle count agent, and (4) a vehicle classifier agent. The MPEG-4 agent and the stationary-vehicle detection agent [2] are well tested

Table 1: Running times of the task allocation

Case	Executed by	PC JDK 1.4	PC JamVM	SmartCam JamVM	SmartCam Native/C++
1a	Partial Allocations (6 Ag.)	20 ms	14 ms	79 ms	13 ms
1b	Partial Allocations (3 Ag)	14 ms	7 ms	55 ms	9 ms
2a	Merge Solutions (6 Ag)	766 ms	4.852 ms	21.363 ms	2.360 ms
2b	Merge Solutions (3 Ag)	9 ms	5 ms	31 ms	4 ms
3a	Overall allocation determination (6 Ag)	786 ms	4.866 ms	21.442 ms	2.373 ms
3b	Overall allocation determination (3 Ag)	23 ms	12 ms	86 ms	13 ms
4	Pruning of infeasible allocations (6 Ag)	14 ms	32 ms	172 ms	26 ms

and evaluated agents, however, the vehicle count and vehicle classify agents only simulated real behavior. A total

of six agents have been instantiated from these four classes, where we used three MPEG-agents (as every camera has to

transmit a live video stream), and one agent of every other agent-type.

Table 1 enlists the execution times of the two stages of the task allocation. On the PCs we have used two Java virtual-machines: (a) Sun's JDK 1.4.2, which uses a just-in-time (JIT) compiler, and therefore reaches better performance values, while (b) the JamVM virtual machine2 is designed as an interpreter and therefore requires higher execution times. Lines 1a, 2a, and 3a represent the execution times if all six tasks had to be considered. As line 2a shows, that the merging process, resulting with a total of 2880 feasible task allocations, requires the most amount of time, therefore we have implemented the core calculation functions of steps 1 and 2 also in C++ to achieve acceptable performance. In a second scenario we have removed the MPEG-4 encoding agents from the surveillance cluster, and allocated them statically to the cameras. The results of the dynamic allocation of the resulting three agents is enlisted in lines 1b, 2b, and 3b in table 1.

Finally, we have also evaluated the performance of the task allocation system at increasing system load. Line 4 enlists the times required to check all 2880 allocations for feasibility. These results show, that the system responds in a timely manner to increased system load.

VII. CONCLUSION

In this paper we have presented a resource-aware dynamic task-allocation system targeting embedded smart cameras. Surveillance tasks are not allocated to the smart cameras directly, but to groups of smart cameras, surveillance clusters, within the tasks are allocated by the smart cameras in a distributed manner. We show that the task allocation can be formulated as a distributed constraint satisfaction problem (DCSP) and present a solution for this DCSP. Therefore we combine the results of the DCSP with a cost function to retrieve the optimal allocation of tasks. Finally, we discuss the mobile-agent based implementation of the system and present results achieved by evaluation of the implemented system.

Future work includes (1) the further evaluation of the system, using more complex scenarios, (2) the tighter combination of finding feasible allocations and calculating the costs for an allocation in order to delete expensive allocations at an early stage, (3) the test and evaluation of the system in real-world scenarios, and (4) the integration of learning agents into surveillance clusters, which influence the behavior of the system by adapting the cost function based on previous behavior, events, and actions.

VIII. REFERENCES

[1] B. Abreu, L. Botelho, A. Cavallaro, D. Douxchamps, T. Ebrahimi, P. Figueiredo, B. Macq, B. Mory, L. Nunes, J. Orri, M. J. Trigueiros, and A. Violante. Video-Based Multi-Agent Surveillance System. In *Proceedings of the 2000 Intelligent Vehicles Conference*, Oct 2000.

[2] M. Bramberger, J. Brunner, B. Rinner, and H. Schwabach. Real-Time Video Analysis on an Embedded Smart Camera for Traffic Surveillance. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 174–181, 2004.

[3] M. Bramberger, B. Rinner, and H. Schwabach. An Embedded Smart Camera on a Scalable Heterogeneous Multi-DSP System. In *Proceedings of the European DSP Education and Research Symposium (EDERS 2004)*, Nov 2004.

[4] K.-P. Chow and Y.-K. Kwok. On Load Balancing for Distributed Multiagent Computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(8):787–801, Aug 2002.

[5] T. Mullanix, D. Magdic, V. Wan, B. Lee, B. Cruickshank, A. Campbell, and Y. DeGraw. Reference Frameworks for eXpressDSP Software: RF5, An Extensive, High-Density System. Technical Report SPRA795A, Texas Instruments, April 2003.

[6] M. K. Perdikeas, F. G. Chatzipapadopoulos, I. S. Venieris, and G. Marino. Mobile agent standards and available platforms. *Elsevier Computer Networks*, (31), 1999.

[7] X. Qin, Q. Zhu, and D. Swanson. A Dynamic Load Balancing Scheme for I/O-Intensive Applications in Distributed Systems. In *Proceedinges of the 2003 International Conference on Parallel Processing Workshops*. IEEE, 2003.

[8] N. Suri, P. Groth, and J. Bradshaw. While You're Away: A System for Load-Balancing and Resource Sharing based on Mobile Agents. In *Proceedings of the First IEEE/ACMInternational Symposium on Cluster Computing and the Grid*, pages 470–473, 2001.

[9] W. Wolf, B. Ozer, and T. Lv. Smart Cameras as Embedded Systems. *IEEE Computer*, 35(9):48–53, Sep 2002.

[10] M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):Sep/Oct, 1998.