6

Swarm Intelligence Algorithms for Dynamic Task Reallocation

The resource allocation mechanisms discussed so far in this book have some features in common: they have (some) a priori knowledge about the application load they are managing, and they are executed in a centralised or hierarchical manner. In this chapter, we explore an approach that does not require explicit information about application load, and that is able to make decisions about resource allocation in a distributed fashion. To better motivate such an approach, let us consider a concrete case study.

Multimedia applications such as video and audio processing are among the most communication and computation-intensive tasks performed by current embedded, high-performance and cloud systems. Hardware platforms with hundreds of cores are now becoming a preferred target for such applications [138], as the computational load in video decoding can be parallelised and distributed across the different processing elements on the chip to minimise metrics such as overall execution time, power or even temperature.

An important design constraint in such systems is performance predictability. There is plenty of evidence showing that inconsistent performance in video decoding applications can lead to reduced user engagement [46]. However, the task of predictably manage multimedia load is not trivial. Video decoding execution times vary greatly depending on the spatial and temporal attributes of the video [63]. Furthermore, when decoding multiple streams of live video (e.g., multipoint video conferencing, multi-camera real-time video surveillance, multiple view object detection), the workload characteristics became increasingly dynamic and thus difficult to model. Thus, efficient resource allocation is critical in achieving load balance, power/energy minimisation and latency reduction [43].

Centralised resource management with a master-slave approach, for instance, is a straightforward way to approach this problem, and is probably good enough for small systems, but there are many issues that appear as one scales up the amount of load to be handled [4, 127]. Cluster based resource management techniques have been introduced (e.g., [69]) to overcome the limitations of centralised systems by partitioning the system resources and employing multiple cluster managers. Despite those efforts, the complexity of dynamic applications and the avaliability of distributed large-scale multi-processor systems motivate the investigation of fully-distributed, autonomous self-organising/optimising mechanisms [21, 97, 145]. Such systems should be able to adapt or optimize itself to changing workload and internal conditions and to recover from faults. Many of these systems implement self-management features by autonomously controlling and adapting task allocation and resource management at runtime.

As a basis for a distributed resource allocation approach, this chapter focuses on the behaviour of biological systems. More specifically, we study the swarm intelligence phenomenon, where the individual decisions made by the members of a swarm in a distributed manner can result in a global behaviour that is beneficial to the whole group. By applying such approach to the management of multi-stream video processing load, we hope to show its potential and to hint on its applicability to similar kinds of resource management problems.

6.1 System Model and Problem Formulation

6.1.1 Load Model

We consider a load model (Figure 6.1) consisting of workflows which resemble a container for parallel video stream decoding requests that may arrive at arbitrary times, but respecting a specified inter-arrival time. Such load model is general enough to describe systems handling a time-varying number of parallel video decoding streams. A video stream consists of an arbitrary number of N independent jobs. Each job (J_i) represents a MPEG group of pictures (GoP), and is modelled via a fixed dependency task-graph, and takes the structure defined in Figure 6.1. Each node in the task-graph is a MPEG-2 frame-level decoder task, and has fixed precedence constraint and communication flow shown via the graph edges. A decoder task can only start execution *iff* its predecessor(s) have completed execution and their output data is available. A decoder task τ_i is characterised by the following tuple: (p_i, t_i, x_i, c_i, a_i); where p_i is the fixed priority, t_i is the period, x_i is the actual execution time, c_i is the worst-case computation cost and a_i is the arrival time of the decoder task τ_i. Decoder tasks are preemptive and have a fixed priority. Tasks within a job are assigned fixed mapping and priorities at the start of the video stream; these exact assignments are used for all tasks of all subsequent jobs in a video stream. Tasks of low resolution video streams are given higher priority over high-resolution video streams, to ensure low-resolution video streams have a lower response-time.

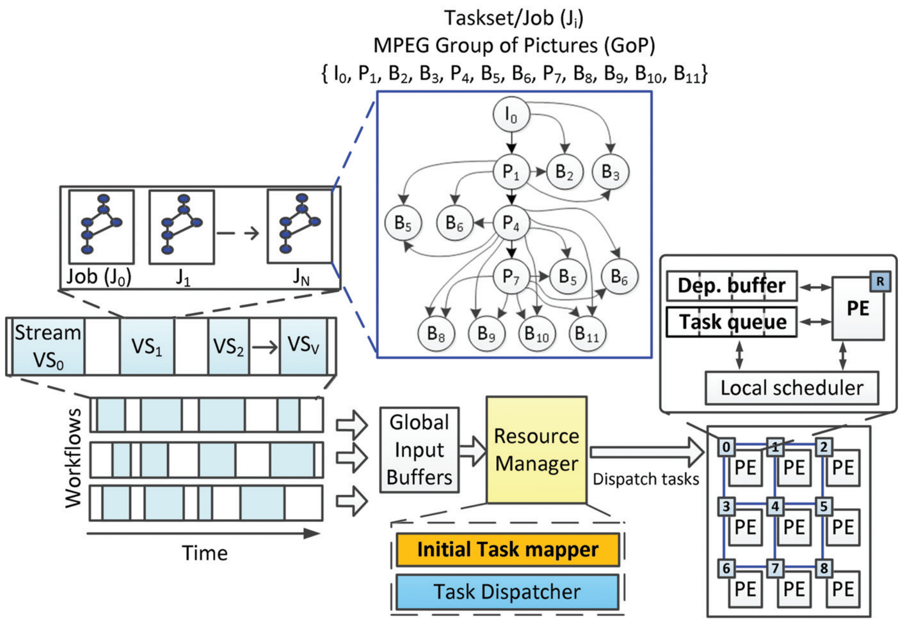

Figure 6.1 System overview diagram.

The spatial resolution of a video stream will correspond directly to the computation cost of the task and the payload of the message flows generated by those tasks. The exact execution time of the tasks are unknown in advance; however, it is assumed that the worst-case computation cost can be estimated. Subtask deadlines are unknown but each job is considered schedulable if it completes execution within its end-to-end deadline ($J_i^r \leq D_{e2e}$). The response-time of a job (denoted J_i^r) is the arrival time of the job to the point in time which all of its subtasks have completed execution. A job is considered late when $(J_i^r - D_{e2e}) > 0$ and late jobs impact the viewing quality of experience (QoE) of the real-time video stream. We assume that the arrival rate of jobs are sporadic, and the arrival pattern of new video decoding streams are aperiodic.

Once a task has completed execution, its output (i.e., the decoded frame data) is immediately sent as a message flow to the processing element executing its successor child tasks, as well as to a buffer in main memory. Message flows inherit the priority of their source tasks, with an added offset to maintain unique message flow priorities. A message flow, denoted by Msg_i is characterised by the following tuple: (P_i, T_i, PL_i, C_i); where P_i is the priority, T_i is the period, PL_i is the size of the message payload and C_i is the maximum no-load latency of message flow Msg_i, which can be calculated a priori and usually depends on the topology of the multiprocessor interconnect and on the total size of the message (i.e., payload plus headers and other overheads).

6.1.2 Platform Model

The multiprocessor platform we target in this chapter is composed of P homogeneous processing elements (PEs) connected by a Network-on-Chip (NoC) interconnect. Each PE has a local scheduler that handles a task-queue which is contained within its local memory. The PEs are directly connected to the NoC switches which route data packets towards any destination PE. We assume the NoC in our platform model uses fixed priority preemptive arbitration, has a 2D mesh topology and uses a deterministic routing algorithm such as in [19].

In such a platform, the no-load latency C_i of a message flow as given in Equation (6.1) includes the hop-distance and the number of data units (i.e., header and payload flits).

$$C_i = (numHops \times arbitrationCost) + (numFlits) \qquad (6.1)$$

We assume that the NoC link arbiters can preempt packets when higher-priority packets request the output link they are using. This makes it easier to predict the outcome of network contention for specific scenarios. We assume all inter-PE communication occurs via the NoC by passing messages. Once a task is released from a global input buffer, it is sent to the task queue of the assigned PE. The PE upon completing a tasks execution, transmits its output to the appropriate PEs dependency buffer. Once a task has completed, the local scheduler picks the next task with the highest priority with dependencies fulfilled, to be executed next. The resource manager (RM) of the system (Figure 6.1), performs *initial task mapping* and priority assignment and *task dispatching* to the PEs. It also maintains a task-to-PE mapping table of the jobs of every admitted and active video stream in the system. The mapping table is essentially a hash-table where keys are task-identifiers and values are node-identifiers. In this chapter, the terms RM and $dispatcher$ are interchangeable as task dispatching is a functionality of the RM. The main responsibility of the RM is to make initial mapping decisions for new video streams, and to dispatch tasks to the mapped PEs according to the task-to-PE mapping table. Most importantly, the system is open-loop as the RM does not gather monitoring information from the PEs.

6.1.3 Problem Statement

In a centralised closed-loop system, PEs would continuously feedback the state of the tasks they were allocated (e.g., their completion time) to a central manager via status message flows. The central manager would then have global knowledge of the system in order to make efficient resource management decisions for future workloads. As discussed in [4, 127], these advantages come at the price of higher communication traffic, congestion hot-spots, higher probability of failure, and bottlenecks around the centralised manager. Furthermore, such issues are made worse as the NoC size and workload increase.

Cluster-based distributed management approaches can offer a certain degree of redundancy and scalability by varying the number of clusters and respectively local cluster managers. However, appropriate cluster size selection is vital to balance communication-overhead/performance; for example, cluster monitoring message flow routes and the cluster manager processing overhead will increase as the cluster size increases. Furthermore, the local cluster managers are still points of failure in the system, where if one of them fails the respective cluster of nodes will severely degrade in performance.

Fully distributed approaches offer higher levels of redundancy and scalability over cluster based approaches for large scale systems, due to not having any central management nodes. However, due to the lack of global knowledge and no monitoring being performed by a centralised authority, the system may be load-unbalanced, and cause jobs to miss their deadlines and become late. To reduce this job lateness of the admitted dynamic varying workload, we follow a bio-inspired distributed task-remapping technique with self-organising properties. This technique builds upon existing bio-inspired load balancing approaches by Caliskanelli et al. [30] and Mendis et al. [91].

6.2 Swarm Intelligence for Resource Management

6.2.1 PS – Pheromone Signalling Algorithm

A distributed load-balancing algorithm based on the pheromone signalling mechanism used by honey bees has been introduced in [30]. That algorithm, henceforth referred to as the *PS algorithm*, was originally developed to improve availability in wireless sensor networks but has features that make it attractive as a general load balancing approach. It is based on the concept of queen nodes (QN) and regular nodes (WN) in a network, drawing inspiration from queen bees and worker bees. The algorithm mimics the process of pheromone propagation by queen bees, which by doing so prevent the birth of new queens. If a queen dies or leaves the hive, the pheromone levels decay and worker bees are then triggered to feed larvae with royal jelly and thus differentiate one of them into becoming the new queen. Perhaps counter-intuitively, the PS algorithm uses the pheromone signalling process to select queen nodes that will be allocated workload (there can be many queens in a system), while worker nodes are not allocated any load unless they become queens themselves (which is a completely different behaviour from the biological system that inspired the approach). QNs are dynamically differentiated from other nodes to indicate they are ready to handle a workload, and the aim of the algorithm is to produces sufficient QNs to handle all the required system functionality.

The algorithm is based on the periodic transmission of pheromone by QNs, and its retransmission by receipients to their neighbours. The pheromone level at each node decays with time and with distance to the source. All nodes accumulate pheromone received from QNs, and if at a particular time the pheromone level of a node is below a given threshold this node will differentiate itself into a QN. This typically happens when this node is too far from other QNs. The PS algorithm consists of three phases, which are executed asynchronously on every node of the network: two of them are time-triggered (differentiation cycle and decay of pheromone) and one of them is event-triggered (propagation of received pheromone).

The first time-triggered phase, referred to as the differentiation cycle (Algorithm 6.1), is executed by every node of the network every T_{QN} time units. On each execution, the node checks its current pheromone level h_i against a predefined level Q_{TH}. The node will differentiate itself into QN (or maintain its QN status) if $h_i < Q_{TH}$, otherwise it will become a WN. If the node is a QN, it then transmits pheromone to its network neighbourhood to make its presence felt. Each pheromone dose hd is represented as a two-position vector. The first element of the vector denotes the distance in hops to the QN that has produced it (and therefore is initialised as 0 in line 4 of Algorithm 6.1). The second element is the actual dosage of the pheromone that will be absorbed by the neighbours.

Algorithm 6.1 PS Differentiation Cycle

```
Input: Differentiation period $T_{QN}$, local pheromone level $h_i$, local threshold
       $Q_{TH}$, initial pheromone dosage $h_{QN}$
Output: Pheromone dose $hd$, Queen Node status $QN_i$
1 while true do
2     if $h_i < Q_{TH}$ then
3         $QN_i$ = true;
4         broadcast $hd = \{0, h_{QN}\}$;
5     else
6         $QN_i$ = false;
7     end
8     wait for $T_{QN}$;
9 end
```

The event-triggered phase of PS deals with the propagation of the pheromone released by QNs (as described above in the differentiation cycle) and received at neighbouring nodes. The purpose of propagation is to extend the influence of QNs to nodes other than their directly connected neighbours. Propagation is not a periodic activity, and happens every time a node receives a pheromone dose. Its pseudocode appears in Algorithm 6.2. Upon receiving a pheromone dose, a node checks whether the QN that has produced it is sufficiently near for the pheromone to be effective. It does that by comparing the first element of the vector hd with a predefined $threshold_{hopcount}$. If the hd has travelled more hops than the threshold, the node simply discards it. If not, it adds the received dosage of the pheromone to its own pheromone level h_i and propagates the pheromone to its neighbourhood. Before forwarding it, the node updates the hd vector element by incrementing the hop count, and by multiplying the dosage by a decay factor $0 < K_{hopdecay} < 1$. This represents pheromone transmission decaying with distance from the source. Figure 6.2 shows four WNs connected to a QN and retransmitting a lower dose of pheromone to their neighbours.

The second time-triggered phase of the algorithm, shown in Algorithm 6.3 is a simple periodic decay of the local pheromone level of each node. Every T_{decay} time units, h_i is multiplied by a decay factor $0 < K_{timedecay} < 1$. It can be easily inferred from the PS differentiation cycle that each node makes its own decision on whether and when it becomes a QN by referring to local information only: its own pheromone level h_i. This follows the principles of swarm intelligence, where decisions are based on local information and interactions within a small neighbourhood.

The computational complexity of the PS algorithm is very low, as each of the phases is a short sequence of simple ALU operations. The communication complexity, which in turn determines how often the PS propagation step

Algorithm 6.2 PS Propagation Cycle

```
Input: Propagation threshold $threshold_{hopcount}$, decay factor $K_{hopdecay}$,
       pheromone dose $hd$
Output: Updated pheromone dose $hd$
1 if $hd$ received then
2     if $hd[1] < threshold_{hopcount}$ then
3         $h_i = h_i + hd[2]$;
4         broadcast $hd = \{hd[1] + 1, hd[2] \times K_{hopdecay}\}$;
5     else
6         drop $hd$;
7     end
8 end
```

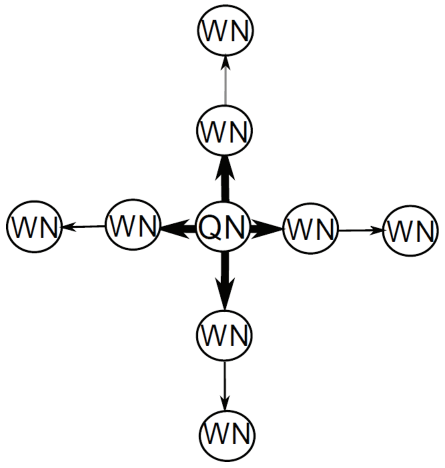

Figure 6.2 PS pheromone propagation

Algorithm 6.3 PS Decay Cycle

```
Input: Decay period $T_{DECAY}$, local pheromone level $h_i$,
decay factor $K_{timedecay}$
Output: Updated local pheromone level $h_i$
1 while true do
2     $h_i = h_i \times K_{timedecay}$;
3     wait for $T_{DECAY}$;
4 end
```

is executed, depends on the connectivity of the network and on the T_{decay} parameter. The protocol also provides a stability property, in that a lone node with no peers will become and always remain a queen node after a given delay, unlike in other distributed resource management approaches where nodes may be probabilistically deactivated for some intervals.

6.2.2 PSRM – Pheromone Signalling Supporting Load Remapping

The PS algorithm described in the previous subsection can be used as a general-purpose load balancing approach. Given a set of parameters, it will converge to a set of QNs that will then serve the system workload as it arrives. Once a QN becomes fully utilized, it can simply decide not to be a QN anymore. By stopping pheromone propagation, its neighbours' pheromone levels will reduce over time due to the decay phase of the algorithm, until one or more will have their levels below the threshold and will differentiate themselves into QNs. The new QNs will then be ready to handle new workload as it arrives.

In this subsection, we explore another possibility: using the PS algorithm to handle load remapping. In this case, it enables distributed resource allocation to optimise a centralised allocation mechanism which may not be fully aware of the state of each resource. Such variation of the PS algorithm has been applied to the problem of dynamically allocating video streams to a NoC-based platform, as described in Section 6.1.

Algorithm 6.4 shows extensions made to the original PS differentiation cycle, aiming to support the remapping functionality. In the original algorithm, Q_{TH} is fixed as a parameter of the algorithm. In this case, Q_{TH} is dynamically adjusted depending on the workload mapped on the resource (namely, the PE connected to the NoC). The *cumulative slack* of the tasks mapped on the PE is used to vary the QN threshold Q_{TH}, such that a node will differentiate itself into a QN if it has enough slack to accommodate additional tasks (line 4). The slack of a task is calculated as the difference between the relative deadline (d_i) and the observed response-time of the task r_i. A negative cumulative slack value indicates the PE does not have any spare capacity to take additional

Algorithm 6.4 PSRM Differentiation Cycle

```
   Input: Differentiation period $T_{QN}$, local pheromone level $h_i$,
   local threshold $Q_{TH}$, initial pheromone dosage $h_{QN}$
   Output: Pheromone dose $hd$, Queen Node status $QN_i$
1  while true do
       /* calc. normalised cumulative TQ slack */
2      $TQ_{Slack} = \frac{\sum_{\forall \tau_i \in PE_{MPT}} (d_i - r_i)}{\sum_{\forall \tau_i \in PE_{MPT}} (d_i)}$;
       /* calc. QN threshold */
3      if $TQ_{Slack} > 0$ then
4          $Q_{TH} = Q_{TH} \times (1 + (TQ_{Slack} \times Q_{TH}^{\alpha}))$;
5      else
6          $Q_{TH} = h_i \times Q_{TH}^{\beta}$;
7      end
       /* determine queen status */
8      if $h_i < Q_{TH}$ then
9          $QN_i$ = true;
10         broadcast $hd = \{0, h_{QN}, QN_{xy}, PE_{MPTinfo}\}$;
11     else
12         $QN_i$ = false;
13     end
14     wait for $T_{QN}$;
15 end
```

tasks, and hence the node is converted or remains a worker node. Line 2 in Algorithm 6.4 shows the calculation of the task queue (TQ) cumulative slack (TQ_{Slack}) of the mapped tasks. If TQ_{Slack} is positive, Q_{TH} is incremented by a ratio defined by $(TQ_{Slack} + Q^{\alpha}_{TH})$; where $\{Q^{\alpha}_{TH} \in \Re \mid 0 \leq Q^{\alpha}_{TH} \leq 1\}$ is a parameter of the algorithm. If TQ_{Slack} is negative, then the algorithm ensures the node does not become a QN in this differentiation cycle, by setting Q_{TH} as a proportion of h_i as given in Line 6; here $\{Q^{\beta}_{TH} \in \Re \mid 0 \leq Q^{\beta}_{TH} \leq 1\}$ is also a parameter of the algorithm. The self-organising behaviour of the distributed algorithm (specifically the Differentiation cycle in Algorithm 6.4), stabilises the number and position of the QNs in the NoC, as time progresses and depending on the workload. A node propagates pheromones immediately after it becomes a queen (line 10). We represent the pheromone dose (hd) as a four position vector containing the distance from the QN, the initial dosage (h_{QN}), the position of the QN in the network (QN_{xy}) and a data structure ($PE_{MPTinfo}$) containing the p_i and c_i of the tasks mapped on the QN. The worker nodes will receive and store this information as the pheromones traverse through the network.

Now that the extension to PS has been introduced, let us focus on its integration to a centralised mapping of video stream tasks. We assume the centralised mapping is performed according to a lowest worst-case utilisation heuristic. Once mapped, a task within a job may be late due to the PE or network route being over-utilised and/or due to the heavy blocking incurred by higher-priority tasks and flows. We then aim to change the task-to-PE mapping of late tasks, such that these causes of lateness can be mitigated. The task-remapping procedure (Algorithm 6.5) is executed by each PE periodically, using only its local knowledge gathered via the pheromone doses.

Algorithm 6.5 illustrates the proposed remapping procedure that utilises the adapted PS algorithm, denoted as $PSRM$. The following steps occur at each remapping cycle (seen in Figure 6.3). Firstly the task with the maximum lateness τ^{MAX}_{L} from the PE task queue, is selected as the task that needs to be remapped to a different PE (line 1). The deadline of a task (d_i) is calculated as a ratio of the end-to-end job deadline (D_{e2e}), as given in [65]. Each node is aware of the nearest QNs (Q_{List}), and their mapped tasks, by storing the information received from each pheromone dose hd. In Lines 3–10, the algorithm evaluates the worse-case blocking that will be experienced for the target task τ^{MAX}_{L} and the number of lower priority tasks that will be blocked, by mapping it onto each $Q_i \in Q_{List}$. Once a list of QNs with lower blocking than the current blocking is obtained (lines 7–9), they are requested (RQ) for their $availability$ (line 11) via a low payload, high priority message flow. The QNs reply (REP) with its availability (i.e., if other worker nodes have been remapped to a QN in that remapping cycle, then the QNs' availability is set to $false$). This avoids unnecessary overloading of QNs. τ^{MAX}_{L} will then be remapped to the

Algorithm 6.5 PSRM Remapping

```
1  while true do
       /* find most late task from task queue */
2      $\tau_i^{MAX\_L} = MAX(\{\tau_i \in TQ \mid (a_i + d_i) \leq t_c\})$;
       /* get current blocking for late task */
3      $B(\tau_i^{MAX\_L}) = getCurrentBlocking(hp(\tau_i^{MAX\_L}))$;
       /* find suitable QNs which offer lower blocking, than
          current blocking */
4      $Q_{List}^B = \{\}$;
5      foreach $Q_i \in Q_{List}$ do
           /* get target task blocking factor */
6          $Self\_B_Q = \sum_{\forall \tau_j \in hp(\tau_i^{MAX\_L})} c_j$;
           /* get number of lower priority tasks */
7          $LP_{size} = \left| lp(\tau_i^{MAX\_L}) \right|$;
8          if $Self\_B_Q < B(\tau_i^{MAX\_L})$ then
9              Insert $\{Q_i, LP_{size}\}$ to $Q_{List}^B$;
10         end
11     end
       /* request for QN availability */
12     $Avlb\_Q_{List}^B = requestAvailability(Q_{List}^B)$;
       /* get available QN that has least amount of lower
          priority tasks */
13     $\{Q_i^{MIN\_LP}, LP_Q^{MIN}\} = MIN(Avlb\_Q_{List}^B)$;
       /* Update dispatcher task-mapping table */
14     Notify dispatcher: $\tau_i^{MAX\_L} \rightarrow PE(Q_i^{MIN\_LP})$;
15     wait for $T_{RM}$;
16 end
```

QN with the least number of lower priority tasks (denoted $Q_i^{MIN\_LP}$) from the available QN list ($Avlb\_Q_{List}^B$) (line 12). Finally, the task dispatcher is notified via message flow to update the task mapping table; the dispatcher looks up the task-id in the table and updates the corresponding node-id with the new remapped node-id. When the tasks of the next job of the video stream arrives into the system they will be dispatched to the node-id indicated by the updated mapping table. Therefore, remapping will only take effect from the subsequent arrival of the next job in the video stream. Even though there is an update message sent to the dispatcher at a remapping event, the remapping decision is achieved purely using local information at each PE, based on the PSRM algorithm.

Figure 6.4 illustrates an example of the remapping procedure in a 4×4 NoC. The (x, y) coordinates refer to the processing node in column x and row y. In step 1 of Figure 6.4, at each remapping interval (T_{RM}) each PE

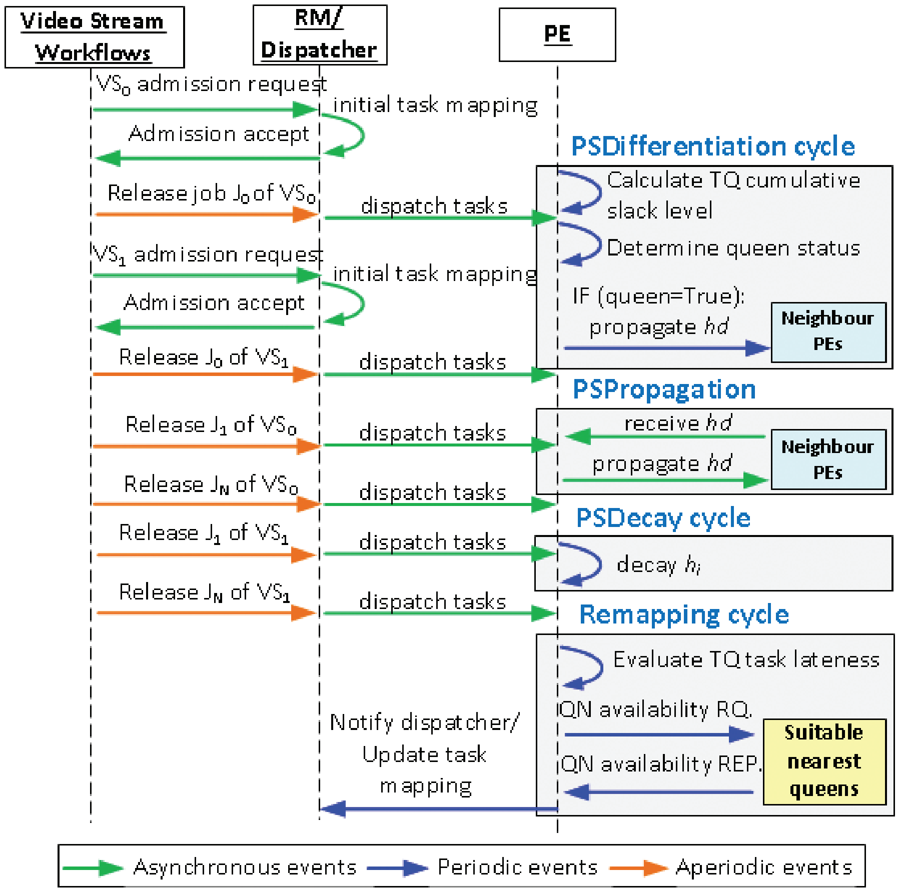

Figure 6.3 Sequence diagram of PSRM algorithm related events. Time triggered (periodic): *PSDifferentiation*, *PSDecay* and *Remapping* cycles; Event triggered: *PSPropagation*.

identifies the late tasks in their task queues; they are also aware of the position of any nearby QNs due to the pheromone signals. τ_1 and τ_2 on PE(1,0) and PE(2,2) are tasks that are late, at that time instant. In step 2, they determine the suitability of each QN to remap the late tasks to. τ_1 can either be remapped to Q(1,1) or Q(3,0); and τ_2 can be remapped on to either Q(3,2) or Q(1,1) but Q(3,2) is not suitable due to the task blocking behaviour and Q(0,3) is not in the Q_{List} due to distance. In step 2, the nodes request for the suitable QNs' availability; in this instance PE(1,0) obtained a lock on Q(1,1) first. Hence, τ_1 will be remapped onto Q(1,1) and τ_2 will be remapped to Q(2,3). In step 3

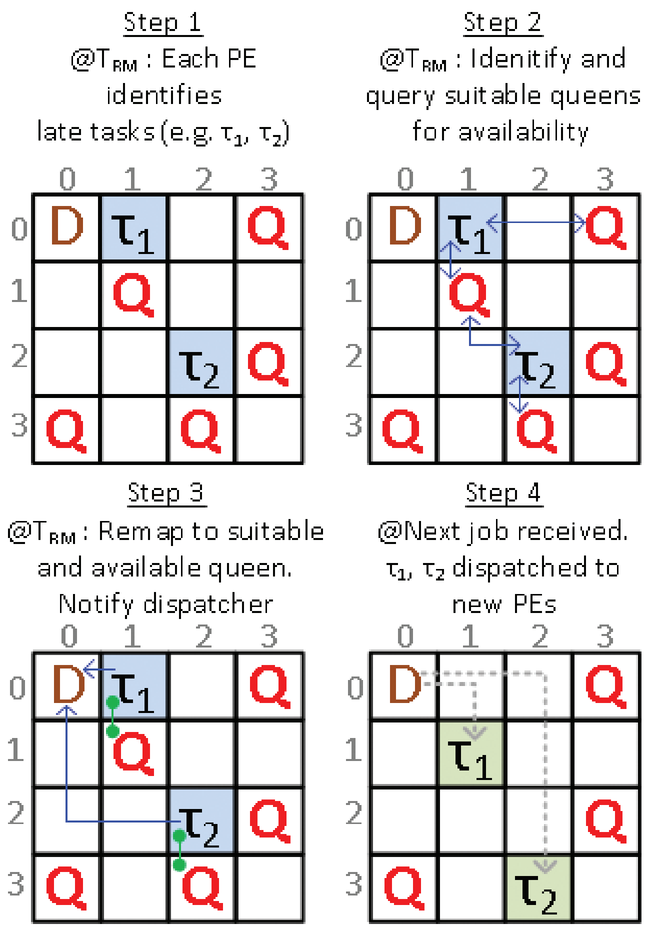
Step 1
@T_RM : Each PE identifies late tasks (e.g. τ1, τ2)
Step 2
@T_RM : Idenitify and query suitable queens for availability
Step 3
@T_RM : Remap to suitable and available queen. Notify dispatcher
Step 4
@Next job received. τ1, τ2 dispatched to new PEs
D
Q
τ1
τ2
0 1 2 3

Figure 6.4 Task remapping example. (Q = queen nodes; D = Dispatcher; $[\tau_1, \tau_2]$ are late tasks; Blue lines represent communication.

the PEs notify the dispatcher via a message flow regarding the remapping. In step 4 the next job arrives and τ_2 and τ_3 are now dispatched to the new processing elements – PE(1,1) and PE(2,3) respectively.

The performance of adaptive algorithms such as PSRM is highly dependent on the selection of a good set of parameters. Manual selection of parameters is not feasible due to the size of the search space. Table 6.1 shows several important parameters obtained via a search-based parameter selection method inspired by [29]. The parameters T_{QN}, T_{DECAY} and T_{RM} and their ratios

Table 6.1 PSRM algorithm parameters

Differenciation cycle (T_{QN})	0.22
Decay cycle (T_{DECAY})	0.055
Remapping period (T_{RM})	6.9
Default QN threshold (Q_{TH})	20
QN threshold inc./dec. factors (Q^{α}_{TH}, Q^{β}_{TH})	0.107, 0.01
Pheromone time and hop decay factors	0.3, 0.15
Pheromone propagation range	3

play a key role in obtaining a good performance from the algorithm. The experimental results during the parameter search process show that the remapping frequency has a significant impact in accuracy and communication overhead. The relationship between these parameters have been investigated extensively in previous work [29, 30]. As a general guideline, to keep the communication overhead low, the event cycles (T_{QN} and T_{RM}) and the QN hormone propagation range must be kept relatively low.

The platform model used in this case study has fixed priority preemptive NoC arbiters and local schedulers. Hence, tasks and flows can be blocked by higher priority tasks and flows. The remapping heuristic takes into account the new tasks' blocking incurred by a possible remapping (lines 4–10 of Algorithm 6.5). However, since the processing nodes lack a global view of the communication flows, the remapping heuristic cannot take into account the change in the overall network *communication interference pattern* caused by the reallocation of the tasks. Therefore, there are situations where remapping a task can result in an actual lateness increase. As shown in Figure 6.3 and Algorithm. 6.4, every T_{QN} time units the worker nodes get updates from all QNs in close proximity to them. However, between subsequent *PSDifferentiation* events, the workload of the QN can change rapidly when the system is heavily utilised, which may lead to inaccurate local knowledge regarding the nearby QNs. Furthermore, late tasks should be remapped ideally before the next job invocation. However, the remapping event is periodic (i.e., every T_{RM} seconds) which allows the remapping overhead to be kept at a minimum, but does not guarantee synchronisation with the workload arrival pattern. Longer periodic events may lead to inconsistency in data and states, but are used to keep the communication overhead at a minimum.

6.3 Evaluation

6.3.1 Experiment Design

To evaluate the resource allocation approach described in this chapter, we performed a number of simulations of realistic load patterns allocated over a 100-core Network-on-Chip platform. A discrete-event, abstract simulator described in [92] has been adopted. The volume of load was configured such that there would be an upper limit of 103 parallel video streams at any given time in the simulation. Experiments were performed under 30 unique workload situations, where the number of videos per workflow, their resolutions and arrival patterns vary based on the randomiser seed used in each simulation run. The computation to communication ratio of the workload was approximately 2:1. The resolution of the video streams were selected at random from a list of low to high resolutions (e.g., from 144p to 720p). The inter-arrival time of jobs in a video stream were set to be between 1 to 1.5 times the D_{e2e}. Tasks were initially mapped to the lowest utilised PE (according to worst-case utilisation) and priority assignment of the tasks followed a scheme were the lowest-resolution tasks get the highest priority. This initial mapping and assignment scheme were constant variables for all evaluations.

6.3.1.1 Metrics

The experiments have multiple dependent variables as described below:

- *Total number of fully schedulable video streams* is the number of all admitted video streams that have no late jobs (i.e., $J_i^r \leq D_{e2e}$).
- *Cumulative job lateness* (C_L^{Jobs}) is calculated as the summation of lateness of all the late jobs from every video stream (v_i) admitted to the system (Equation (6.2)). In Equation (6.2), J_i^L is a late job and VS denotes all the video streams admitted to the system. We measure the job lateness with remapping enabled/disabled, hence a reduced C_L^{Jobs}, when remapping is enabled is considered an improvement to the system. This metric gives us a notion of how the remapping technique reduced the lateness of the unschedulable video streams, which directly affects the QoE of the video stream.
- *Communication overhead* is calculated as the sum of the basic latencies (C_i) of every control signal in the respective remapping technique. In the PSRM algorithm these are the pheromone broadcast and QN availability request signals. In the cluster-based technique the PE status update traffic and the inter-cluster communication traffic contributes to the overhead. Furthermore, the task dispatcher notification messages in all the remapping techniques are included in the overhead. Lower communication overheads lead to less congested networks as well as lower communication energy consumption [39].
- *Distribution of PE utilisation* is calculated by the measured total busy time for every PE on the network during a simulation run. PE utilisation gives a notion of the workload and a lower variation in workload distribution is desirable. Overloading a single resource and/or having

a high number of idle PEs, are undesirable properties which may lead to reduced reliability and increased wear-and-tear.

$$C_L^{Jobs} = \sum_{\forall v_i \in VS} \left[\sum_{\forall J_i^L \in v_i} (J_i^r - D_{e2e}) \right] \tag{6.2}$$

$$\text{Comms. overhead} = \sum_{\forall msg_i \in ControlMsgs} C_i \tag{6.3}$$

6.3.1.2 Baseline Remapping Techniques

The PSRM resource manager was evaluated against the following baselines:

- $CCPRM_{V2}$ – is a cluster-based management proposed in [91] as an improvement of the original work by Castilhos et al. [33]. It is configured with a cluster size of 2 × 5 (i.e., 10 clusters).
- *Centralised management* – is essentially CCPRM$_{V2}$ with only one 10 × 10 cluster. A single centralised resource manager receives status updated from every slave PE in the network and performs periodic remapping as described in [91]. The manager notifies the task dispatcher of any remapping decisions.
- A *random remapper* – is a remapping scheme where, every remapping interval each PE selects the most late task in its task queue and randomly selects another node on the network to remap to. The task dispatcher is notified of the remapping event.

6.3.2 Experimental Results

6.3.2.1 Comparison between clustered approaches

The comparison of CCPRM$_{V1}$ and CCPRM$_{V2}$ for the C_L^{Jobs} metric is shown in Figure 6.5(a). In this plot a positive improvement indicates that task remapping has helped to reduce the cumulative job lateness in the admitted video streams. A negative improvement indicates that the remapping has instead worsened the lateness of the jobs. Each sample in the distribution corresponds to a simulation run with a unique workload. It is clear that the modifications made to the original CCPRM$_{V1}$ technique has resulted in an improvement in reducing job lateness. In CCPRM$_{V1}$ a majority of the data shows negative improvement, while CCPRM$_{V2}$ shows more positive job lateness improvement. However, this improvement has costed a 4% increase in communication cost. Certain constraints in the local remapping decisions in CCPRM$_{V2}$ would result in more communication with neighbouring clusters which might explain the increased overhead.

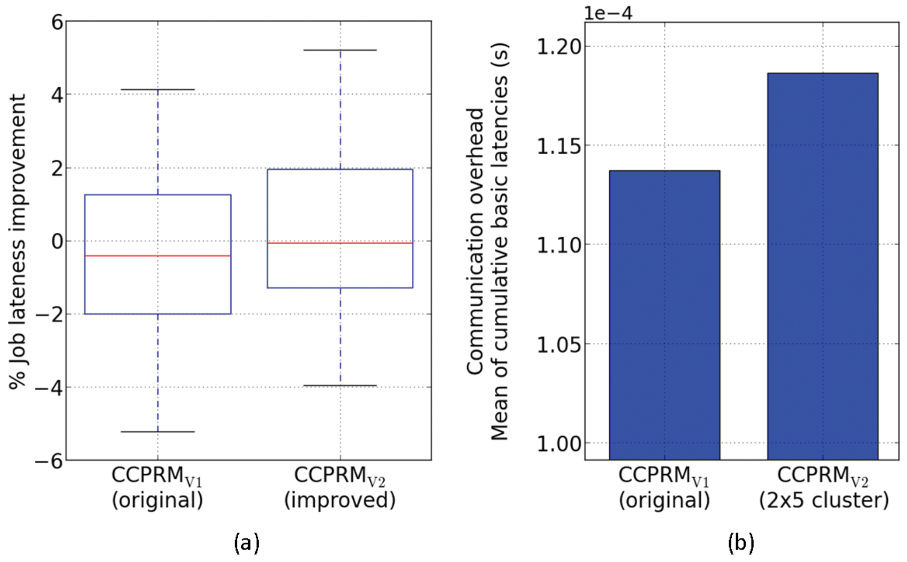

Figure 6.5 Comparison of CCPRM$_{V1}$ (original) and CCPRM$_{V2}$ (improved). (a) Cumulative job lateness improvement. (b) Communication overhead.

6.3.2.2 Comparison regarding video processing performance

Figure 6.6 shows the distribution of cumulative job lateness improvement for each of the remapping techniques. Firstly, all the techniques show both negative and positive improvements; hence, under certain workload situations the remapping techniques have failed to improve the lateness of the jobs. However, a majority of the distribution in both PSRM and CCPRM$_{V2}$ are in the positive improvement region. PSRM has a smaller spread in lateness compared to the baselines. The upper quartile and a significantly large proportion of the inter-quartile range (IQR) falls in the positive improvement area, which is not seen in any of the baselines. In over 60% of the workload scenarios PSRM will produce positive improvement to the job lateness of the video streams but the actual improvement is small (up to 3%–4%). Futhermore, in Figure 6.7, we can see PSRM is marginally better than the CCPRM$_{V2}$ in the number of fully schedulable video streams. CCPRM$_{V2}$ shows a better job lateness improvement over the centralised management, because the monitoring traffic is shorter in route-length and hence is less disruptive to the data communication. We can see that the centralised management has the highest number of schedulable video streams out of the evaluated remappers. This could indicate that CCPRM$_{V2}$ and PSRM gave significant job lateness improvements only to a few video streams while the centralised management was able to make minor improvements to multiple video streams. The random remapper shows

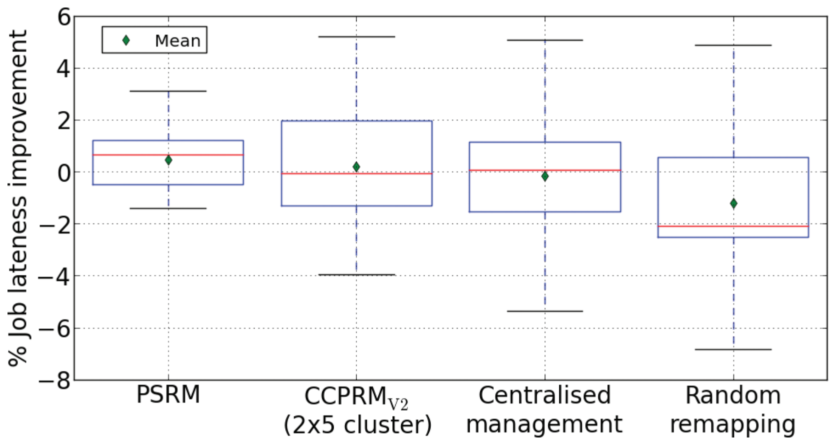

Figure 6.6 Distribution of cumulative job lateness improvement after applying remapping.

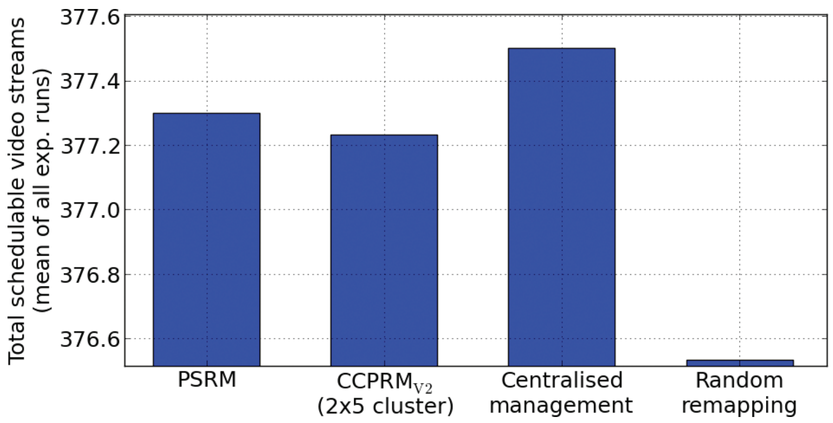

Figure 6.7 Comparison of fully schedulable video streams for each remapping technique.

the worst results with a majority of the experiments resulting in negative improvements and produces the lowest number of fully schedulable video streams. It was interesting to note that there were a few scenarios where random remapping produced significant job lateness improvements, which is seen by the high upper whisker in the box plot (Figure 6.6).

6.3.2.3 Comparison regarding communication overhead

PSRM shows a significant communication overhead reduction when compared to the baselines (Figure 6.8). The mean and IQR of PSRM communication overhead is lower than the baselines but the larger variance of the

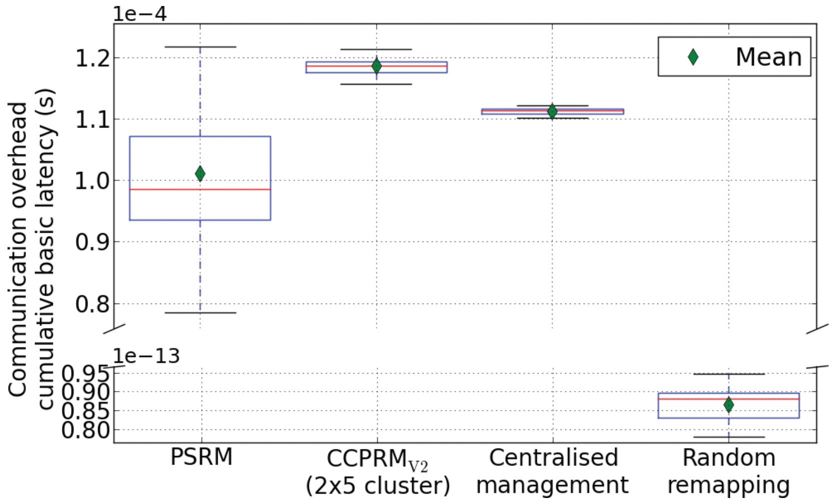

Figure 6.8 Communication overhead of the remapping approaches.

results is due to the different range of workloads and their effect on the QN differentiation cycle in each experimental run. The maximum overhead is comparable to that of CCPRM$_{V2}$. Both the CCPRM$_{V2}$ and centralised management show a higher and narrower distribution of communication overhead than PSRM. A higher upper whisker in PSRM shows that under certain workload scenarios the overhead can be costly and similar to the CCPRM$_{V2}$ baseline. The lower communication overhead distribution of the centralised manager when compared with CCPRM$_{V2}$, is due to the lack of inter-cluster communication. In the centralised management scheme communicating tasks mapped at the middle of the NoC will suffer due to the network congestion caused by the incoming monitoring traffic. Furthermore, these traffic flows will occupy longer routes than CCPRM$_{V2}$. Furthermore, we are shown in [69], that the centralised managers' communication overhead issues become severe after the NoC size exceeds 12×12. The random mappers' communication overhead is many orders of magnitude lower than the others as it only incurs overhead when notifying the task dispatcher regarding remapping decisions.

6.3.2.4 Comparison regarding processor utilisation

The PE utilisation distribution shown in Figure 6.9(a), indicates the PEs with higher utilisation using lighter shade, while the darker shades show PEs with low utilisation levels. The data shown in this plot are normalised such that each remapping technique is relative to each other. PSRM shows a slightly similar variation in the workload distribution to CCPRM$_{V2}$ with only a single

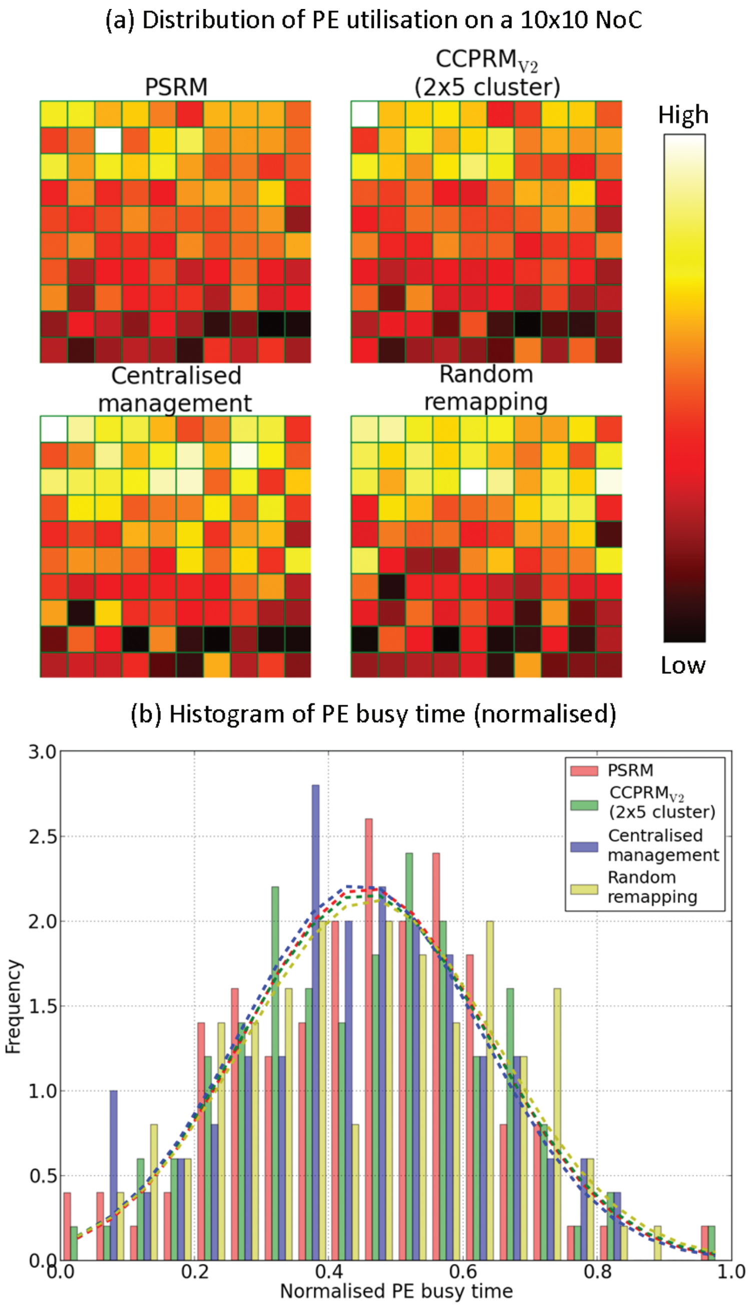

Figure 6.9 Comparison of PE utilisation for all remapping techniques. (a) Distribution of PE utilisation across a 10 × 10 NoC. (b) Histogram of PE busy time (normalised; 20 bins).

PE with extremely high utilisation and a few with very low utilisation. The curves fitted to the histogram data shown in Figure 6.9(b) indicates that all four remapping techniques have a similar spread of workload distribution. However, closer examination to the statistical properties of the distributions (given in Table 6.2), indicate that centralised management has the lowest variance and the mean utilisation. The random remapper has the highest variance and mean utilisation. The frequency spikes of the centralised and random remappers in Figure 6.9(b) at 0.8, 0.4 and 0.7 probably give rise to these statistical properties. PSRM shows a higher mean utilisation and lower distribution variance when compared with CCPRM$_{V2}$.

6.3.3 Outlook

Overall the results indicate the PSRM technique helps to reduce lateness in the video stream jobs and to increase the number of schedulable video streams when compared with the CCPRM$_{V2}$ remapper. It is important to note that this improvement, even though is marginal, comes at a much lower communication overhead (up to 30% lower than the cluster-based and centralised approaches). A higher maximum lateness improvement can be obtained using CCPRM$_{V2}$, but only in 40%–50% of the workload scenarios. Communication overhead of CCPRM$_{V2}$ may grow as the cluster sizes increase, however in the PSRM technique this overhead will vary depending on the distribution of QNs in the network. Also, unlike in the baseline approaches, in PSRM the pheromone signalling message paths are usually short (only a few hops) regardless of the NoC size increases. A centralised resource manager can help to evenly distribute the workload much better than PSRM, because of its global knowledge of the PE status and the mapped tasks. However, PSRM shows better workload distribution when compared with a cluster-based approach. Unlike in the cluster-based management, in PSRM, there are no resource managers in the network; each node executes a simple set of rules using only local knowledge to collectively improve the performance. The execution cost of the remapping event (Algorithm 6.5) is bounded by the number of QNs in the local vicinity and the number of tasks mapped on the node. In the cluster

Table 6.2 PE utilisation distribution statistics. Lower variance (var.) = better workload distribution

	mean	var.
PSRM	0.455	0.033
CCPRM$_{V2}$	0.454	0.034
Centralised management	0.447	0.032
Random remapper	0.462	0.035

based approach each local processor's execution overhead for management functions (such as remapping, inter-core communication, monitoring etc.) would increase as the cluster size increases. Unlike in the centralised approach, PSRM is decentralised hence has no single point of failure or an isolated communication congestion area. Each PE has the capability of performing remapping and becoming a QN, hence the level of redundancy in the system is greater than in the baseline remappers.

One of the identified limitations in PSRM is the sensitivity of the parameters. The parameters need to be tuned for a specific network size and can produce varying results based on the nature of the workload. Parameters that are suitable for a smaller NoC size may not necessarily produce favourable results for a larger NoC. The experimental results during the tuning of the parameters for the PSRM and CCPRM$_{V2}$ techniques show that the remapping frequency has a significant impact on the performance and communication overhead. Furthermore, depending on the value of T_{QN} and T_{DECAY}, PSRM will vary in the time it takes to identify suitable QNs in the network, and hence better performance results can be seen after longer runs of the algorithm.

6.4 Summary

This chapter presented extensions to a fully distributed resource management technique based on swarm intelligence. We have shown how such an approach can be applied to a multiple video stream decoding application with several unknown dynamic workload characteristics, on a NoC-based multicore. With low communication overhead, it relies on task-remapping strategies to progressively distribute the workload in the network and to reduce the overall job lateness.

The experimental results have shown that the bio-inspired remapper gives a marginal (2%–4%) improvement in lateness reduction but incurs 10%–30% lower communication overhead and minor improvement to workload distribution than the baseline cluster-based and centralised management schemes. The centralised management allows the system to increase the number of total schedulable video streams, but the improvement to the cumulative job lateness of the late video streams is poor and the communication overhead is higher than in the proposed technique. The benefits of the centralised management degrade as the scale of the network and workload increase [4]. Results show that the proposed PSRM approach give a marginal benefit in reducing the cumulative job lateness of the video streams when compared against the CCPRM$_{V2}$ cluster based resource management approach; however it is important to note that the improvement is obtained using a significantly less (up to 30% lower) communication overhead than the cluster based approach.

A reduced communication overhead may lead to lower energy consumption [39] and less congested communication network, making PSRM more efficient than the cluster-based approach. Furthermore, unlike in the centralised or cluster-based approaches the proposed PSRM remapping technique does not depend on a single or group of management entities. Each node is independent and capable of relocating late tasks to improve the overall job latency, hence adopting this technique introduces a high degree of redundancy for NoC-based multi/many-cores that require reliable and timely operation.