Optimal Assignment of a Tree-Structured Context Reasoning Procedure onto a Host-Satellites System

Hailiang Mei, Pravin Pawar, Ing Widya
University of Twente
Dept. of Computer Science
P.O. Box 217, 7500 AE, The Netherlands
{H.Mei, P.Pawar, I.A.Widya@utwente.nl}

Abstract

In this paper, we study the problem of an optimal assignment of a tree-structured context reasoning procedure onto the computation resources in a host-satellites configuration. The objective function to be minimized is the end-to-end processing delay, which is a crucial factor in a number of context- aware applications, e.g. mobile healthcare applications. The presented solution is a modification of an earlier method proposed by Bokhari, in which the optimal assignment problem to minimize the bottleneck processing time is transformed into a path-searching problem in a doubly weighted graph. Due to the incompatible requirements raised in our study, e.g. a-prior known location of the sensors, we propose a colouring scheme and a new search algorithm in this paper to obtain the optimal assignment in order to satisfy our objective.

1. Introduction

A context-aware application adapts according to the context changes such as user context and communication and computation environment context. To achieve the adaptive behavior, it relies on a process to convert a lower level context information (possibly obtained from multiple sensors) into a higher level context information in the form understandable by the context-aware application [1, 2]. We refer to this process as *context reasoning procedure*.

A criteria of interest to perform the context reasoning procedure could be to *minimize the end-to-end context processing delay* and thus to facilitate instantaneous application adaptation to the context changes. The following example from the healthcare domain will be helpful to understand this concept (Figure 1). In the *context-aware epilepsy tele-monitoring application* [3, 4], a patient's mobile terminal (e.g. a PDA) is connected with a number of sensor boxes. Each sensor box can measure and process a different type of context information, e.g. patient's ECG and patient's activities (*lower level context*). The processed results are sent to the mobile terminal to further calculate the probability of an epileptic seizure (*higher level context*). If a serious seizure attack is detected or forecasted, the mobile terminal sends an alarm to a healthcare center automatically. Based on this alarm, the healthcare center could inform both the patient and the caregivers in the vicinity of the patient to take appropriate actions. Undoubtedly, the earlier the warning is received, the better the chances are to avoid catastrophic consequences.

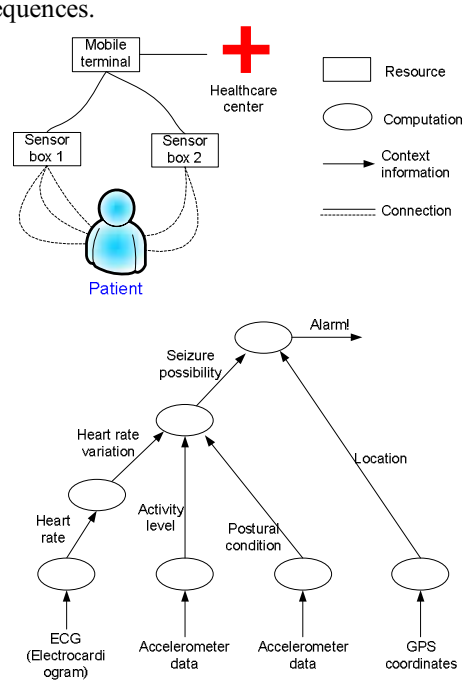

Figure 1: An epilepsy tele-monitoring application

Considering the heterogeneity of the communication and computation resources involved in the context-aware applications, a challenge is to investigate on how to minimize the end-to-end processing delay in the context reasoning procedure by effectively utilizing the available resources. This is a problem of *assigning* distributed computation across the resources of a networked multi-

1-4244-0910-1/07/$20.00 ©2007 IEEE

processor system [5]. The assignment of a context reasoning procedure onto the networked computation resources is found to be similar to an industrial case studied earlier by Bokhari [6] in the area of parallel computing. In order to find an *optimal assignment* of the tree-structured task graph onto a host-satellites system to minimize the bottleneck processing time, Bokhari presented an elegant solution to construct a *doubly weighted assignment graph*. In this assignment graph, any path connecting two distinguished nodes represents a possible assignment and *S* weight and *B* weight of this path can be calculated based on the two weights of each edge along the path. Therefore, the assignment problem becomes a path-searching problem which can be solved by the *SB* (to find a path with minimal of max(***S*** weight, ***B*** weight)) algorithm presented in [6].

However, the addressed context-aware applications, in particular the context reasoning procedures, have incompatible requirements than the conditions needed by the method proposed by Bokhari. In the pre-described tele-monitoring application, for example, the number of sensor boxes and the sensors connected to these boxes are a priori known. Therefore, sensors are not freely assignable to sensor boxes as would be required if the method of Bokhari will be used. Furthermore, our different objective function requires the study of a new measure of the path in the assignment graph. In this work, we modify Bokhari's method to our needs, in particular we propose a colouring scheme to solve the sensor connections and replace the *SB* algorithm by the *SSB* algorithm (to find a path with minimal of ***S***um (***S*** weight, ***B*** weight)) that is used to obtain the optimal assignment with minimum end-to-end processing delay.

The rest of this paper is organized as follows: Section 2 briefly summarizes the related work and highlights the difference between our problem and the problem considered by Bokhari. Section 3 formulates the problem of assigning a context reasoning procedure optimally across a host-satellites network. Section 4 proposes the algorithm to search an optimal *SSB* path in a *Doubly Weighted Graph* (DWG). Section 5 explains how to transform the targeted assignment problem into a *SSB* path-searching problem in a coloured DWG and how the proposed *SSB* algorithm can be applied in this coloured graph. Section 6 concludes the work.

2. Related work

The research on the *task assignment* (also known as partitioning or mapping) was pioneered by Stone and Bokhari [7, 8] in the early 80's. For most of the cases, it is a well-known NP-complete problem [9]. Therefore, researchers further focused on *heuristic approaches* to solve these problems [10-12]. In his inspiring work [6], Bokhari studied several mapping problems with special structures and proposed the exact algorithms with the polynomial time complexity, e.g. *chain to chain mapping* and *tree to host-satellites mapping*. Improved algorithms on *chain to chain mapping* have been reported in [13-17]. Improved algorithms on *tree to host-satellites mapping* have been reported: In [14], a different implementation of Dijkstra's algorithm is used to reduce the complexity of the SB algorithm from $O(|V|^2\log|V|)$ to $O(|V|(\log|V|)^2)$, where $|V|$ denotes the number of tasks in the tree. In [18], a *tree condense* procedure is introduced to reduce the size of the task tree. The condensed task tree has a monotonic structure which permits a faster search ($O(|V|\log|V|)$) of an optimal assignment without the help of doubly weighted assignment graph..

Our work (in tree to host-satellites mapping) differs from Bokhari's original approach and those follow-up studies in two aspects.

Firstly, the following two constraints [6, 18] which do not hold for our case are relaxed by a *colouring scheme*: 1) If two nodes are assigned to a satellite, their lowest common ancestor is also assigned to the same satellite. 2) There are as many satellites as there are leaf nodes in the tree and that it is possible not to use them if the optimal assignment so dictates, i.e. partitioning on the tree is done first and then the leaf nodes are located on the satellites based on the result.

Secondly, Bokhari proposed the *SB* algorithm to find a partition that minimizes the bottleneck processing time while our goal is to find a partition that minimizes the end-to-end processing delay. We propose the *SSB* algorithm to tackle this different objective.

3. Problem formulation

Based on the described tele-monitoring example and other observations (e.g. examples from the SNMP based network monitoring, industry cases presented by Bokhari), a context reasoning procedure could be modeled as a tree consisting of a number of CRUs (*Context Reasoning Units*) (Figure 2). A CRU is defined as a unit of context reasoning procedure which takes care of one of the functions involved in the reasoning of a higher level context from the lower level context. A directed link represents the precedence relation of CRUs, i.e. the flow of context information. The raw context information is captured by sensor nodes, i.e. a kind of CRU at the leaf level which does not perform any context processing. The ultimate reasoning is performed by the CRU at the root node, the result of which is used by the context-aware application to achieve the desired behavior. In many cases, the computation resources needed to execute the context reasoning procedure can be modeled as a star network, i.e. a single *host machine* connecting to a number of *satellites*. For example, in the tele-monitoring example (Figure 1),

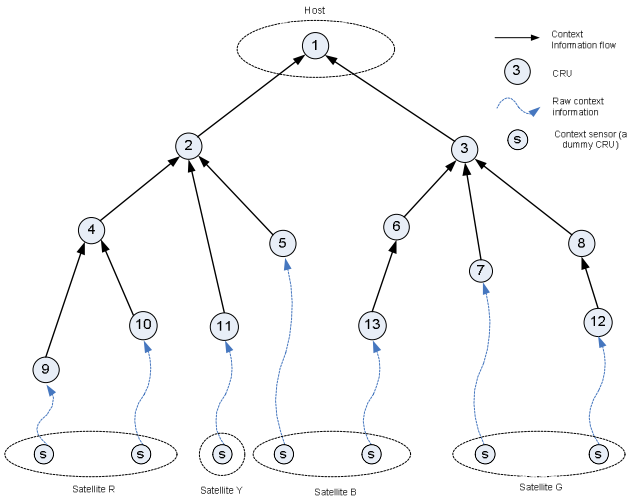

Figure 2: A CRU tree where some sensors are physically linked to the same satellite

the sensor boxes are satellites while the mobile terminal is a host.

We assume that the CRUs placed on the host cannot start processing unless they receive the processed context information from all the precedent CRUs located on the satellites. Therefore, we formulate our problem as follows: *Given a context reasoning procedure modeled as a tree structure and the corresponding computation and communication resources modeled as a host-satellites network, find an optimal assignment of CRUs to the host and satellites such that it results in a minimum end-to-end processing and communication delay, i.e. to minimize the summation of maximum processing time spent at the satellite (including the time to transmit context from the satellite to the host) and the processing time required at host machine to obtain the higher level context.*

4. Optimal *SSB* path in a doubly weighted graph

In this section, we study a doubly weighted graph (DWG) and introduce a measure of paths in the DWG, the *SSB* weight. An algorithm is proposed to search for the optimal path that has minimum *SSB* weight.

4.1. Doubly weighted graph (DWG) and *SSB* weight

Similar to [6], DWG $G=(V,E)$ has two ordered non-negative weights associated with each edge e of E, for example: a *sum weight* $\sigma(e)$ and a *bottleneck weight* $\beta(e)$. We define further an S weight and a B weight of a path P that connects two distinguished nodes in G as $S(P)$ and $B(P)$ respectively, which are defined as:

$$S(P)= \Sigma[\sigma(e_i)] \text{ and } B(P)= \max[\beta(e_i)] \text{ where } e_i \in P$$

Now we introduce the measure of the *SSB* (**S**ummation of ***S*** weight and ***B*** weight) weight of a path P as $SSB(P)$ where $SSB(P)=\lambda \cdot S(P) + (1-\lambda) \cdot B(P)$. λ is the *weighting coefficient* between the S weight and B weight and its value is between 0 and 1. The optimal *SSB* path(s) in a

```
Function SSB(G(V,E): a doubly weighted graph, lambda):path;
var optimal_SSB_path := NULL;
var optimal_SSB_weight := infinite;
var S_weight := 0;
var B_weight := 0;
All edges' S weight are multipled by lambda;
All edges' B weight are multipled by (1-lambda);
G':=G;
Find p, the shortest S weight path in G';
B_weight := p.B_weight;
WHILE (G' is connected & optimal_SSB_weight > S_weight)
        update G' by removing all the edges whose bottleneck weight >= B_weight;
        IF (p.SSB_weight < optimal_SSB_weight)
                optimal_SSB_weight := p.SSB_weight;
                optimal_SSB_path := p;
        ENDIF
        Find new p, the shortest S weight path in G';
        S_weight := p.S_weight;
        B_weight := p.B_weight;
ENDWHILE
return optimal_SSB_path;
```

Figure 3: *SSB* algorithm

doubly weighted graph is defined as the path with minimum *SSB* weight. This *SSB* weight is therefore a different measure compared to the *SB* weight studied by Bokhari [6], where the *SB* weight of a path P is defined as $\max(S(P),B(P))$.

4.2. Algorithm for finding the optimal *SSB* path

Inspired by the earlier discussions on DWG [6, 19], we present the *SSB* algorithm for finding the optimal *SSB* path. This algorithm works by recording the candidate optimal *SSB* paths and progressively eliminating edges from the graph when they cannot be a part of the optimal *SSB* path, until the graph becomes disconnected or further searching will definitely not yield any better path.

Given a DWG G_0, the goal of the *SSB* algorithm is to iteratively find an optimal *SSB* path P_{opt} connecting two distinguished nodes, e.g. "S" and "T". Prior to the start, two state variables are initiated: the candidate optimal *SSB* path P_{can} is set as *NULL* and the candidate's *SSB* weight SSB_{can} is set as $+\infty$.

In the ith iteration, the algorithm starts with the searching of path P_i in G_{i-1} with the minimum S weight. A number of shortest path-searching algorithms can be applied for this purpose, e.g. Dijkstra algorithm. We compare $SSB(P_i)$ to the SSB_{can}. If $SSB(P_i)$ is smaller than SSB_{can}, we store P_i into P_{can} and store $SSB(P_i)$ into SSB_{can}; Otherwise, P_{can} and SSB_{can} are kept the same. Then, the edges in $E_i=\{e_i|e_i\in G_{i-1}, \beta(e_i)\geq B(P_i)\}$ are removed from G_{i-1} to yield a reduced G_i which will be used in the next iteration. The reason that we can safely remove E_i is because all the edges in E_i except for those that belong to P_i are not part of the optimal *SSB* path. Therefore, at the end of this iteration, either G_i contains the optimal *SSB* path of G_0 or P_{can} is the optimal *SSB* path.

The iteration ends until either the new G_i becomes disconnected or the S weight of P_i is greater than the current SSB_{can}, i.e. all the remaining paths' *SSB* weights are greater than SSB_{can}.

Now, the algorithm found the optimal *SSB* path in G_0, $P_{opt}=P_{can}$.

Each iteration in the algorithm applies a shortest path searching with the complexity of $O(|V|^2)$, where $|x|$ denotes the cardinality of x [20]. In the worst case, $|E|$ times of iteration are required, i.e. eliminating one edge per iteration. Thus, the total time complexity of this algorithm is $O(|V|^2|E|)$. The pseudo code of this algorithm is presented in Figure 3.

An example of finding an optimal *SSB* path by applying the proposed algorithm is illustrated in Figure 4. Given this simple DWG, three iterations are executed to identify an optimal *SSB* path (<5,10>-<5,10>) with *SSB* weight of 20.

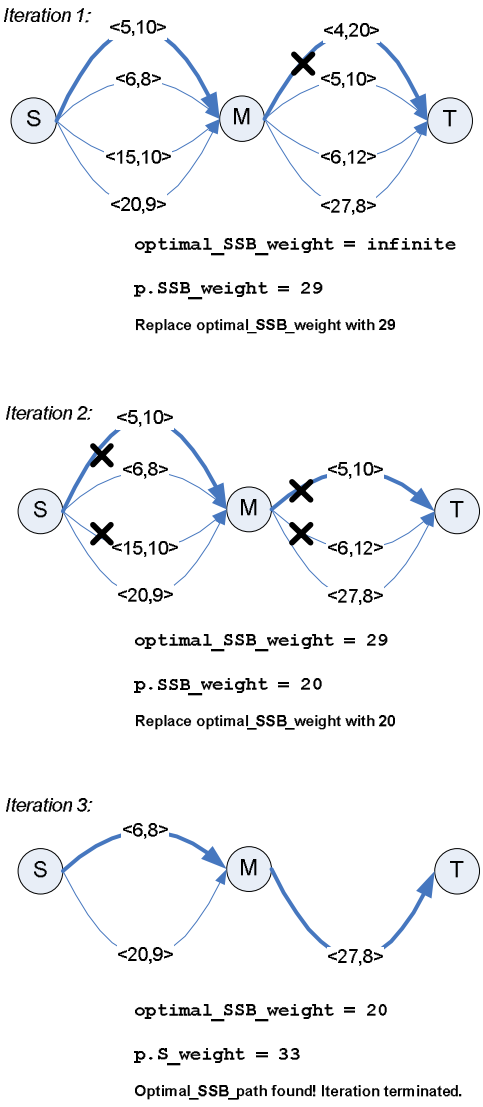

Figure 4: An example of searching for the optimal *SSB* path, where the thick path indicates the newly found shortest path based on sum weight and the cross indicates the paths to be eliminated.

5. Optimal assignment of CRUs on a host-satellites network

In this section, we present the step by step solution of finding the optimal assignment of CRUs to the execution nodes in the following sections.

5.1. Colouring the CRU tree

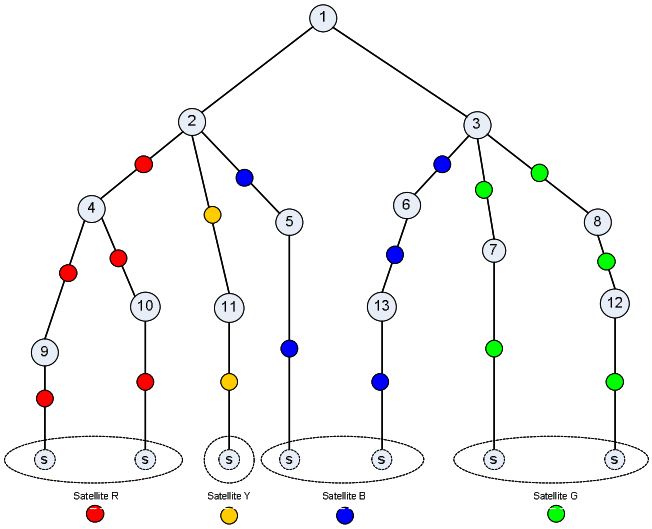

Figure 5: Colouring the edges of the CRU tree

First, we paint each satellite with a distinguishable colour, e.g. Red for satellite R, Yellow for satellite Y, Blue for satellite B and Green for satellite G. Then, each edge of a CRU tree is painted by "propagating" the colour of satellite towards the root node (Figure 5). The exceptions to this colouring are the edges of $<CRU_1, CRU_2>$ and $<CRU_1, CRU_3>$ since the propagated colours conflict. This phenomenon implies that CRU_1, CRU_2 and CRU_3 have to be deployed on the host. It is because that they need to process the context information obtained from the multiple satellites.

5.2. Building the coloured assignment graph

Similar to Bokhari's approach, all the sensor nodes are merged into a single dummy node "Δ". The nodes which will constitute to the assignment graph (squares in Figure 6) are inserted in each face of the CRU tree and on the left and right-hand sides of the tree. An assignment graph of this modified tree is now drawn by adding an edge between every pair of nodes that belong to faces that have a common coloured tree edge. This assignment graph is actually kind of dual graph or dual network [21] of the "closed" CRU tree. The edge of this assignment graph inherits the colour of the tree edge it crosses. This procedure and the resulting coloured assignment graph are shown in Figure 6.

5.3. Labelling the assignment graph

In this section, we follow the same approach of Bokhari [6] to doubly weight each edge e of the assignment graph with two ordered non-negative weights,

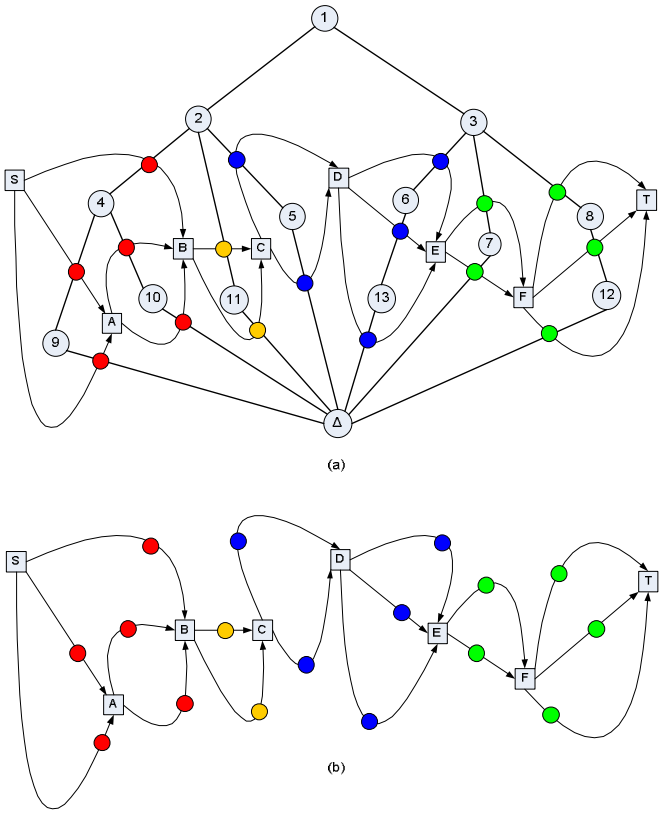

Figure 6: (a) illustrates the procedure of building the coloured assignment graph; (b) presents the assignment graph by eliminating the original tree.

i.e. a *sum weight* $\sigma(e)$ and a *bottleneck weight* $\beta(e)$, such that the corresponding computation and communication time required by the host and satellite are represented by these weights respectively.

For every CRU, there are two possible locations to execute it: the *host* or its *correspondent satellite*. For example, in Figure 2, if the sensors connected to CRU_5 and CRU_{13} are physically linked to satellite B; then B is called CRU_5 and CRU_{13}'s correspondent satellite. Thus, for CRU_i to process one frame of context information, two processing time indicators are available: the required processing time on its correspondent satellite s_i and the required processing time on the host h_i. These two values can be obtained by using the analytical benchmarking or task profiling techniques [22]. For every known context reasoning procedure, the data exchange between the two connected CRUs can be known a priori. Based on the amount of data exchanged and the approximate characteristics of the communication link between the host and satellite, it is also feasible to determine the time required to transfer the context information (i.e.

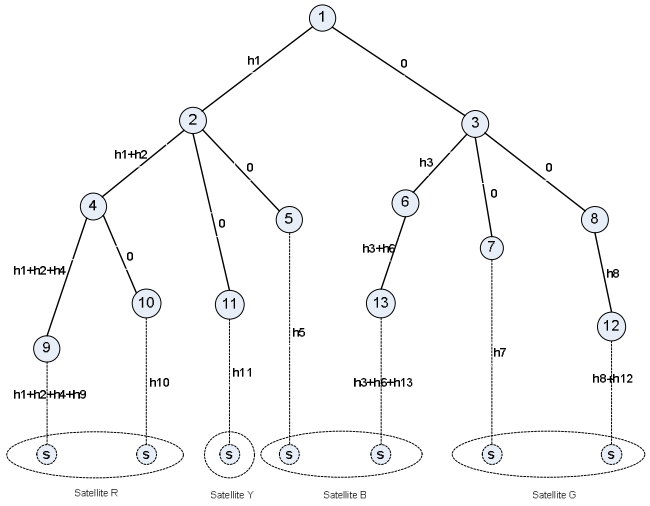

Figure 8: Assigning host weights to tree edges

communication cost). We define $c_{i,j}$ as the time required to transfer one frame of context data from CRU_i to CRU_j. In case the raw context information (from sensors) is transmitted over the communication link, $c_{s,i}$ is used to denote the time required to transfer one frame of raw context data to CRU_i.

In order to finalize the coloured DWG, the bottleneck weight (β) is added first by considering the required processing time on satellites. Suppose an edge in the assignment graph cuts the CRU tree into two parts: The upper part contains among others the original root CRU and the lower part contains a set of CRU(s) named as τ. Then, the β weight of this edge is the sum of all s_i for all $CRU_i \in \tau$ and the communication cost resulting from this cut. As explained earlier in this section, s_i represents the time required for CRU_i to execute on its correspondent satellite. For example, consider the assignment graph edge <*D*,*E*> that crosses CRU tree edge < CRU_3, CRU_6> in Figure 6, the β weight on this edge is $s_6+s_{13}+c_{6,3}$. Another example is that the β weight on the edge *A*-*B* that crosses tree edge <Δ, CRU_{10}>, equals to $c_{s,10}$.

Secondly, we add the sum weight (σ). First label the host execution time on all the edges in the original CRU tree as shown in Figure 8: Give all edges connecting parent CRUi to child CRUj an initial cost w_{ij}=0. Traverse the nodes of CRU tree from the root in pre-order. When visiting node CRU_j which has parent CRU_i and leftmost child CRU_k give edge <CRU_i, CRU_k> the weight $w_{jk}=w_{ij}+h_j$. An exception is the left-most edge leaving root node and its weight is h_1. Then each edge of the assignment graph is given a σ weight equal to the weight of the CRU tree edge that it crosses. For example, the assignment graph edge *S*-*B* crossing CRU tree edge <CRU_2, CRU_4> is given a σ weight of h_1+h_2 (c.f. Figure 6(a)).

5.4. Finding the optimal *SSB* path

Now, we have built a coloured doubly weighted assignment graph (Figure 6 b). Each path connecting the nodes S and T in this graph corresponds to a partition of the CRU tree on the host and satellites. The end-to-end processing delay of this partition equals to the path's *SSB* weight, i.e. the summation of the coloured path's *S* weight and *B* weight. The coloured path's *S* weight is defined in the same way as the non-coloured DWG, i.e. $S(P)=\Sigma[\sigma(e_i)]$. The coloured path's *B* weight is defined as *the maximum among the summations of the bottleneck weights per colour*:

$$B(P) = \max_{e_i \in P}[\sum_{red} \beta(e_i), \sum_{yellow} \beta(e_i), \sum_{blue} \beta(e_i), ...]$$

In order to identify the optimal assignment, it is required to search for the coloured path with minimal *SSB* weight. The previously proposed *SSB* algorithm can be adapted in two aspects to serve this purpose (Figure 10).

Firstly, a specific feature in this coloured DWG (Figure 6 b) is that the path with minimum *S* weight is always on the top of the assignment graph. This is because of the property that a partition on the top indicates that the minimum number of CRUs is deployed on the host; thus implies the least total processing time at the host. Therefore, it is possible to skip the step of shortest-path searching in the *SSB* algorithm.

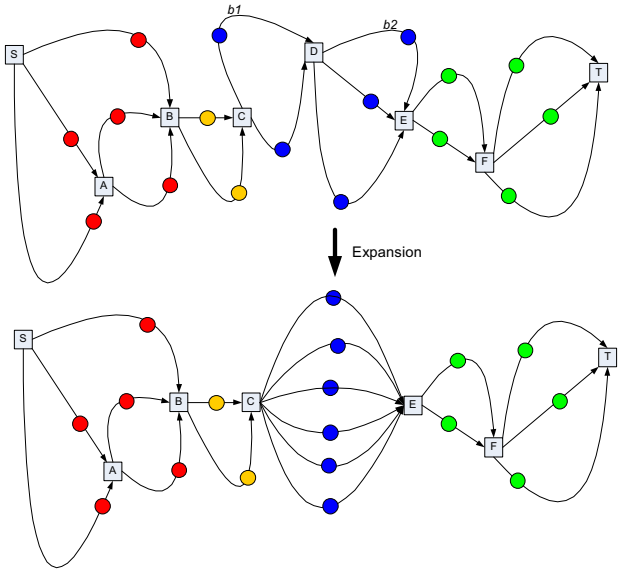

Figure 9: Expanding part of the assignment graph

Secondly, an exception should be taken into account in the iterative steps of removing edges with a larger bottleneck weight: when the *B* weight of the shortest-path (determined using *S* weight) is contributed by the

```
Function coloured_SSB(G(V,E): a coloured doubly weighted graph, lambda):path;
var optimal_SSB_path := NULL;
var optimal_SSB_weight := infinite;
var S_weight := 0;
var B_weight := 0;
All edges' S weight are multipled by lambda;
All edges' B weight are multipled by (1-lambda);
G':=G;
Find p, the shortest S weight path in G';
B_weight := p.B_weight;
WHILE (G' is connected & optimal_SSB_weight > S_weight)
        IF (p.B_weight is a summation of several edges in the same colour)
                update G' by expanding that part of graph;                    //graph expansion
        ENDIF
        update G' by removing all the edges whose bottleneck weight >= B_weight;
        IF (p.SSB_weight < optimal_SSB_weight)
                optimal_SSB_weight := p.SSB_weight;
                optimal_SSB_path := p;
        ENDIF
        Find new p, the shortest S weight path in G';
        S_weight := p.S_weight;
        B_weight := p.B_weight;
ENDWHILE
return optimal_SSB_path;
```

Figure 10: Adapted *SSB* algorithm for coloured DWG

subsequent edges having the same colour, that part of the assignment graph should be expanded (refer to "expansion" step in Figure 10) before any edges are eliminated. This ensures that during the edge removal step, only the edges which do not contribute to the optimal *SSB* path anymore are removed. For example, as shown in Figure 9, if the *B* weight of the shortest path (the topmost path) is a sum of the bottleneck weights of the two blue edges (labeled *b1* and *b2*), the entire blue part of the graph should be expanded into a number of edges, each of which represents a possible path between node *C* and *E*.

Therefore, in this specific case, due to the skipping of shortest-path searching step and the graph expansion, the running time of the adapted *SSB* algorithm is in the order of $O(|E'|)$, where $|E'|$ is the number of edges in the expanded graph.

6. Conclusion and future work

This paper addresses the problem of minimizing the context processing delay in a context-aware application by optimally utilizing the heterogeneous computation resources. Bokhari has studied problems with similar structure in the area of parallel computing. Inspired by his pioneering approach, we also transform our assignment problem into a path-searching problem. Due to the physical dependency between the task and the resources which is not considered by the earlier work, we propose a colouring scheme. Furthermore, as our context processing criteria is different, we propose the *SSB* algorithm for the new objective function. This *SSB* algorithm can find the path corresponding to the optimal assignment which minimizes the end-to-end processing delay. The complexity of this *SSB* algorithm is of order $O(|V|^2|E|)$ in general. In our specific constructed coloured DWG, the adapted *SSB* algorithm is of the order $O(|E'|)$, where $|E'|$ is the number of edges in an expanded assignment graph.

We are aware of the existence of applications that do not fall into the model we considered. Therefore, we plan to address a more general model, i.e. DAG-tasks-to-DAG-resources assignment problem. Since, very likely, no algorithms with polynomial time complexity will be found to solve the general problem, our future work will be focused on heuristic approaches, e.g. *Branch-and-Bound* [23] and *Genetic Algorithms* [24].

Acknowledgement

This work is part of the Freeband AWARENESS Project. Freeband is sponsored by the Dutch government under contract BSIK 03025. (http://awareness.freeband.nl)

References

1. Henricksen, K., et al. *Middleware for Distributed Context-Aware Systems*. in *International Symposium on Distributed Objects and Applications (DOA)*. 2005.

2. Dey, A.K., G.D. Abowd, and D. Salber, *A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications.* Human-Computer Interaction, 2001. **16**: p. 97-166.

3. Halteren, A.v., et al., *Mobile Patient Monitoring: The MobiHealth System.* The Journal of Information Technology in Healthcare, 2004. **2**(5).

4. Tönis, T., H. Hermens, and M. Vollenbroek-Hutten, *AWARENESS D4.18, Context aware algorithm for discriminating stress and physical activity versus epilepsy*. 2006.

5. Norman, M.G. and P. Thanisch, *Models of machines and computation for mapping in multicomputers.* ACM Computing Surveys, 1993. **25**(3): p. 263-302.

6. Bokhari, S.H., *Partitioning problems in parallel, pipelined, and distributed computing.* IEEE Transactions on Computers, 1988. **37**(1): p. 48-57.

7. Stone, H.S., *Multiprocessor scheduling with the aid of network flow algorithms.* IEEE Transactions on Software Engineering, 1977. **3**: p. 85-93.

8. Bokhari, S.H., *On the Mapping Problem.* Computers, IEEE Transactions on, 1981. **C-30**(3): p. 207.

9. Garey, M.R. and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979: Miller Freeman, San Francisco.

10. Eshaghian, M.M. and Y.C. Wu. *Mapping heterogeneous task graphs onto heterogeneous system graphs*. in *Heterogeneous Computing Workshop, 1997. (HCW '97) Proceedings., Sixth*. 1997.

11. Lo, V.M., *Heuristic algorithms for task assignment in distributed systems.* Computers, IEEE Transactions on, 1988. **37**(11): p. 1384.

12. Cooper, K., et al. *New grid scheduling and rescheduling methods in the GrADS project*. in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. 2004.

13. Anily, S. and A. Federgruen, *Structured partitioning problems.* Operations Research, 1991. **13**(1): p. 130-149.

14. Hansen, P. and K.-W. Lih, *Improved Algorithms for Partitioning Paroblems in Parallel, Pipelined, and Distributed Computing (Correspondence).* IEEE Transactions on Computers, 1992. **41**(6): p. 769-771.

15. Woeginger, G.J. *Assigning chain-like tasks to a chain-like network*. in *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. 2001.

16. Olstad, B. and F. Manne, *Efficient Partitioning of Sequences.* IEEE Transactions on Computers, 1995. **44**(11): p. 1322-1326.

17. Khanna, S., S. Muthukrishnan, and S. Skiena, *Efficient Array Partitioning.* Automata, Languages and Programming, 1997: p. 616-626.

18. Ashraf Iqbal, M. and S.H. Bokhari, *Efficient algorithms for a class of partitioning problems.* Parallel and Distributed Systems, IEEE Transactions on, 1995. **6**(2): p. 170.

19. Christofides, N., *Graph theory: An algorithmic approach.* 1975: Academic Press.

20. Edmonds, J. and R.M. Karp, *Theoretical improvements in algorithmic efficiency for network flow problems.* Journal of ACM, 1990. **19**(2).

21. Ahuja, R.K., T.L. Magnanti, and J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. 1993: Prentice Hall; United States Ed edition.

22. Maheswaran, M. and H.J. Siegel. *A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems*. in *In Seventh Heterogeneous Computing Workshop. IEEE Computer Society Press*. 1998.

23. Satyanarayanan, M., *The many faces of adaptation.* IEEE Pervasive Computing, 2004.

24. Wang, L., et al., *Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach.* Journal of Parallel and Distributed Computing, 1997. **47**: p. 8-22.

Biography

Hailiang Mei received his BSc degree in 2001 from Beijing University of Technology, China and his MSc degree from the Delft University of Technology, Netherlands in July, 2003. Both are in Electrical Engineering. Since July 2005, he works towards his PhD degree in the Architecture and Services of Network Applications group (ASNA) of the Computer Science Department of the University of Twente, Netherlands. His PhD topic is on Smart Distribution of Bio-Signal Process in Mobile Healthcare. His current research interests include task assignment, dynamic reconfiguration and

component based software engineering. He is a student member of IEEE.

Pravin Pawar received his M. Tech. degree in Computer Science and Engineering from Indian Institute of Technology, Bombay, India in January 2002. After serving for a few years in IT industry and academics, since June 2005 he works as a PhD candidate in the Architecture and Services of Network Applications group of the Computer Science Department of the University of Twente, Netherlands. His PhD work consists of providing context-aware computing support for the nomadic mobile services hosted on resource constrained devices. His other research interests include mobile computing, artificial intelligence and mobile e-commerce. He is a student member of IEEE and IEEE Communication Society.

Ing Widya is Assistant Professor of the Faculty of Electrical Engineering, Mathematics & Computer Science at the University of Twente, the Netherlands. He is a member of the Architecture and Services of Network Applications group (ASNA) and his research is embedded in the projects of the Centre for Telematics and Information Technology (CTIT). He received his PhD in Signal Processing and his current research interests cover enterprise modelling of communication supports for networked applications, design and analysis of application-context aware services and protocols operating over large-scale infrastructures like the Internet, including QoS and standardised multimedia format encodings.