Power Conservation Strategy for Mobile Computers Using Load Sharing

Mazliza Othman Stephen Hailes
M.Othman@cs.ucl.ac.uk S.Hailes@cs.ucl.ac.uk
University College London, Department of Computer Science, London, U.K.

Power management is an important aspect of mobile computing. Previous works on power conservation have concentrated on the hardware approach. In this paper, we propose a different approach of power conservation strategy for mobile computers which is based on the concept of load sharing. User jobs are transferred from a mobile host to a fixed host to reduce power consumption by the CPU. Simulation results show that under suitable conditions, transferring job can extend battery lifetime by up to 20%. Transferring jobs to a fixed host does not only extend battery lifetime but also gives users access to faster machines, hence improving job response time.

I. Introduction

Advancing technology in wireless communication will allow roaming users to access the network while away from their office. An example of an application for roaming users is the MOST project [3], where engineers take their laptops with them to the field. Wireless Coyote [9] is an example of how wireless technology can be used in an educational product. The Wireless Coyote experiment involved four groups of students who used the product on a field trip. The application allowed the groups of students in different locations to share the data they collected in real-time.

Power management has become one of the important issues which need to be addressed in order to support roaming users. A mobile computer operates on battery power. Under continuous use, the battery will last for about 2 - 3 hours. Ideally it should last for 8 hours (one working day). Since the projection on progress in battery technology shows that only a 20% improvement in battery capacity will occur over the next 10 years [17], it is vital that power utilisation is managed efficiently and economically.

This paper discusses an approach to reduce power consumption by the CPU. The CPU consumes approximately 31% power on a mobile computer [8]. The proposed approach borrows from the concept of load sharing. Jobs are migrated from a mobile host to a fixed host for execution in order to reduce power consumption by the mobile's CPU.

Executing the jobs at a fixed host will not only reduce power consumption but will also give the mobile access to a faster machine and, thus, improves performance.

Previous works on power conservation have concentrated on the hardware approach. [4] and [6] discuss a strategy of spinning down the hard disk during idle periods which will reduce power consumption on mobile computers. [10] discusses a dynamic disk spin-down algorithm which receives input from a set of experts. Each expert is assigned a weight which is updated after each trial depending on how accurate or misleading the expert's prediction is.

[5] discusses a storage alternative, the flash memory, which consumes less power, has low latency and has high throughput for read accesses.

[24] proposed a method of power savings by reducing the CPU clock speed. In this paper, Weiser et al shows that is better to spread work out by reducing clock speed and voltage than to run the CPU at full speed for short bursts and then idle. This approach stems from the non-linear relationship between clock speed and energy saving.

The rest of this paper is organised as follows. Section 2 gives some background on load sharing and mobile computing issues. Section 3 discusses the algorithms, the simulation model, experiments and assumptions. Section 4 discusses the results and finally, section 5 concludes.

II. Background

A. Load Sharing in Distributed Systems

Load balancing or load sharing is a strategy to distribute workload among processors in a distributed system. Some literature distinguishes between load balancing and load sharing. Load balancing is often defined as a strategy which attempts to assure that each processor in a system has equal load. Load sharing, on the other hand, is usually referred to as a strategy which attempts to share loads in a distributed system without attempting to equalise its load. Both strategies have the same goal, which is to make better use of the system resources (usually the CPU) by making sure that no nodes are idle.

In the context of this study, we shall use the term load sharing. A load sharing algorithm consists of 2 policies. A *transfer policy* decides when a job should be transferred. This is usually determined based on the number of jobs in the queue waiting to be serviced. The *location policy* decides to which host a job should be transferred. This is done either by choosing a host randomly or by using workload information. The workload information may be obtained either by probing a subset of hosts or by collecting the information periodically. If the information is collected periodically, an optimal period has to be determined. Collecting the information frequently will result in accurate and up to date information but will incur more overhead. On the other hand, a less frequent period will result in out of date information being used.

Several load sharing algorithms have been proposed. The algorithms vary from those which make no use of system state information (e.g. Random algorithm of [7]) to those which attempt to make use of global state information (e.g. [15]).

Not all jobs are suitable for migration. [26] observes that some jobs are immobile, i.e. they have to be executed locally. Examples of such jobs are those which perform local services

and/or require local resources.

Migrating a job involves packaging it at the source node, transmitting it through the communication network and unpacking it at the destination. A number of previous works assumed that the probing cost and/or network cost to be negligible ([22], [7], [18] and [16]). Probes are usually assumed to take zero time. Other works which made a non-negligible delay assumptions were [11], [20], [25] and [14]. If a file used by a job has to be transferred to be migrated along with it, the delay factor will be even more significant.

This study investigates the use of load sharing for a different purpose. We are investigating if load sharing is effective as a power conservation strategy for mobile computers. The CPU is one of the components which consumes significant battery power. In this study, we migrate jobs from a mobile computer to a fixed host and by doing so, we hope to reduce power consumption by the CPU and thus, extends battery lifetime.

B. Some Mobile Computing Issues

The idea behind mobile computing is to allow users wireless access to the network regardless of their location. Users will want to run the same applications that they run on a fixed network. They will expect the same computational environment regardless of their location [21]. Alonso and Korth envisioned that applications run will be similar to those currently served by laptop computers and these applications will demand various transaction and transaction-like services [1].

Ideally, users should receive the same quality of service and performance as if they were connected to the wired network. This, however, is not possible due to the limitations of the wireless networks, among which are limited bandwidth, variability in the quality of wireless connection, lower memory capacity and limited battery power. Although the limitations are being addressed in successive technologies, the performance disparity between the wireless and wired networks is likely to remain [23].

Mobile hosts have significantly lower memory capacity and computing power compared to a fixed host. For this reason, [2] proposed that the computation and communication load should be borne by the static part of the network to the extent possible. Doing so will reduce the burden of computation on mobile hosts and also helps conserve battery power.

Transmitting a message from a mobile host consumes more power than receiving a message of the same size. In order to conserve power message transmission from a mobile host should be kept to a minimum. Communication in a cell needs to be asymmetric to reduce power consumption on the mobile hosts and to better exploit the broadcast capability of the medium [2].

Power can also be conserved by design and efficient operation. Applications can conserve power by reducing their appetite for computation, communication and memory [8].

The power conservation strategy we are proposing takes on the suggestion that the computation load should be borne by the fixed network by [2].

C. Load Sharing in Wireless Networks

Mobility introduces new challenges as a lot of assumptions made regarding distributed networks are no longer valid in wireless networks. Wireless networks are associated with low bandwidth, high delays and frequent disconnection. This section discusses how the assumptions made regarding load sharing policies in a mobile environment will differ from those in distributed networks.

Previous works on load sharing focused on job migrations on fixed networks. The location policy has to select a suitable host for the job transfer. In a wireless network, the mobile support station (MSS) (or a base station) is the only fixed host a mobile host can communicate with directly. Therefore, jobs can only be migrated from the mobile to the MSS. The location policy is only needed in cases where the MSS decides to delegate transfer requests to other fixed hosts. In this case, any of the existing location policy may be used.

The MSS may transfer the migrated job to another fixed host. This however, can be made transparent to the mobile host. Where a job is actually executed is not important so long as the mobile gets the result back when it needs it.

Due to the limited bandwidth inherent in wireless network, it is no longer valid to assume negligible communication delays. Probes can no longer be assumed to take zero time. The delays incurred in transmitting a probe and receiving a reply have to be taken into consideration.

In this study, we assume that only newly arrived jobs may be considered for remote execution or transfer. When a new job arrives, the load sharing algorithm will determine whether the benefit of migrating a job outweighs the cost of migrating it. If the benefit outweighs the cost, the mobile host will send a transfer request to the MSS. The MSS will then send a reply accepting or rejecting the request. When a MSS accepts a transfer request, it implicitly agrees to execute the job even if the mobile moves to a new cell.

Since the main goal of migrating jobs is to conserve battery power on mobile hosts, jobs should only be migrated if the amount of power consumed migrating the job is less that the amount of power consumed by the CPU if the job is to be executed locally.

III. Experiments

A. The Algorithms

We ran the simulations using three different algorithms. The algorithms perform the same calculations but differ in the way they estimate the CPU time required to execute a job.

If a job is migratable, the algorithms carry out the following calculations :

1. the amount of power consumed by the CPU if the job is to be executed locally,
2. the amount of power consumed for transmitting and receiving messages if the job is to be transferred to a fixed host,
3. an estimate of job response time for local execution vs. remote execution.

The messages exchanged between the mobile and the MSS in order to execute the job remotely are :

- a transfer request sent by the mobile host to the MSS,

- a reply from the MSS accepting or rejecting the request,
- the job transfer itself,
- execution result returned by the MSS.

If the calculation shows that power consumed by (2) is less than (1) and that remote execution will not result in higher response time, the mobile will send a transfer request to the MSS. In the transfer request, the mobile also specifies the maximum period of time it is willing to wait for the MSS to execute the job on its behalf. The maximum waiting time is based on the time required to execute the job. It is imposed in order to avoid the mobile from having to wait too long for the remote execution which will cause a degradation in job response time.

The MSS decides if it will be able to execute the job within the specified time based on the execution time of jobs it has waiting to be executed in its queue. If the MSS is able to do so within the specified time, it will send a reply accepting the request. Otherwise, the request will be rejected and the mobile host will execute the job locally.

As was mentioned earlier, the algorithms differ in the way that they estimate the CPU time required to execute a job. How each algorithm estimates the CPU requirement is explained below.

Algorithm 1 : Optimal Load Sharing (LS)

This algorithm assumes a priori knowledge of CPU requirement. The CPU time required to execute a job used by this algorithm is obtained from the trace data. This value is used in calculation (1) and as the maximum waiting time. The calculation performed by this algorithm always gives accurate estimates and it is therefore, used as an upper bound algorithm.

In real life, it is unlikely that this information will be available in advance. Algorithm 2 and algorithm 3 make no assumption of this a priori knowledge. The CPU time used in calculation (1) is an average value.

Algorithm 2 : History

The simulation is first run in a no load sharing mode (NLS) and the average CPU time taken to execute each job is calculated. For example, the CPU time taken for each execution of job A is totalled and the average value is calculated at the end of the simulation. Later, when the job is run in load sharing mode using the History algorithm, the average value calculated in the previous simulation run is used in calculation (1). This value is also used as the maximum waiting time.

Since the same trace data were used when running the simulation in NLS and LS mode, the users' workload is exactly the same. In reality it is highly unlikely that the same workload will be reproduced. Therefore, this algorithm can be considered to be an upper bound for the use of history information.

Algorithm 3 : Adaptive Load Sharing (ALS)

The adaptive algorithm learns and adapts its decision based on previous execution of jobs. It works as follows. When job A is executed for the first time, its CPU requirement is unknown and therefore, no assumption can be made regarding the feasibility of transferring it. Job A will be executed locally and the algorithm will keep a record of the CPU time taken to execute the job.

The next time job A is executed, the CPU time from the previous execution is used in calculation (1) to estimate if remote execution will be beneficial. Each time job A is executed, the CPU time taken to execute the job is used to calculate a new average value which will be used in future calculations. As in the History algorithm, this average value is also used as the maximum waiting time.

We expect this algorithm to be the most practical compared to the other two algorithms since it makes no assumption of a priori knowledge and is able to adapt its behaviour according to a user's working pattern.

B. The Simulation

The simulation is written using Maisie [19] which is a C-based simulation language. Entities are defined to represent mobile hosts, mobile support stations, fixed hosts and communication channels. The entities communicate using message passing.

C. Trace Data

We would like the simulation to represent the type of workload generated in real life. For this reason, trace data for the simulation were collected from Sun workstations in the undergraduate labs at the Computer Science Department, UCL. Process accounting was used to collect trace data for 24-hour periods.

Since the trace data were collected from undergraduate labs, the type of jobs run were mostly text processing, program compilation, email, web browsers etc. We tried to obtain some trace data for multimedia applications. Unfortunately that was not possible because the multimedia applications run on Solaris machines while process accounting can only run on Sun machines.

We summarised the data to contain only the time period when the machines were being used. The time period when no user was logged on were deleted. This gave us trace data for a period of 4 - 8 hours. Each job type is classified either migratable or non-migratable. Examples of migratable jobs are program compilation, simulations and program runs, while examples of non-migratable jobs are interactive jobs, text formatting and email.

Among the information provided by the trace data are job name, job start and end time and the CPU time (in seconds) taken to execute a job.

Even though the trace data were not collected from mobile applications, this is not an unreasonable approximation. It is plausible to assume that mobile users will use their mobile computers as an extension to they way it is currently used in the work place instead of for totally different type of applications.

D. Power Consumption

An AST Power Executive 325/SL NiMh battery provides (14.4 V * 2.4 A-hr) $\approx$ 34.6 W-hr. Transmitting and receiving will consume 3.4W and 1.7W respectively [12]. [8] lists power consumption of hardware components of a portable computer. Based on this information, the battery will last for about 3.4 hours, assuming that general power consumption (i.e. power consumption by the basic components such as the display, hard drive, keyboard etc) will be about 10.1 W.

Power consumed when transmitting =

$$3.4 \cdot \frac{t_r}{3600} \text{W-hr}$$

t_r = time taken to transmit a message (sec)
Power consumed when receiving is calculated in a similar way.
Power consumed by the CPU is =

$$P_{CPU} \cdot \frac{t_m}{3600} \text{W-hr}$$

P_{CPU} = power consumed by the CPU
t_m = CPU time taken to execute a job on a mobile (sec)
Based on the information provided in Intel's Application Note [13], the average active CPU power consumption is assumed to be 4.59 Watts and idle power consumption, i.e. doze mode, (with Advanced Power Management) is 1.24 Watts.

E. Assumptions

At the moment the simulation only considers job migration between a mobile and its Mobile Support Station (MSS) or Base Station (BS). The parameters used in order to determine if it is worth migrating a job are :

- available bandwidth;
- job size;
- power consumed by the CPU to execute a job on the mobile host;
- power consumed transmitting and receiving messages.

B = available bandwidth
R_1 = packet size of request message
R_2 = packet size of transfer reply
J = size job to be transferred + data
R_3 = packet size of the returned result
P = power remaining on the mobile host
P_t = power consumed transmitting packet(s)
P_r = power consumed receiving packet(s)
P_{CPU} = power consumed to execute a job on the mobile host

$$\text{cost of sending transfer request} = \frac{R_1}{B} \cdot P_t$$

$$\text{cost of receiving a reply} = \frac{R_2}{B} \cdot P_r$$

$$\text{cost of transferring a job} = \text{cost of transmitting a job} + \text{cost of receiving the result} = \frac{J}{B} \cdot P_t + \frac{R_3}{B} \cdot P_r$$

$$\text{cost of migrating a job} = \text{cost of sending a request} + \text{cost of receiving a reply} + \text{cost of transferring a job} =$$

$$\alpha \left(\left(\frac{R_1}{B} \cdot P_t \right) + \left(\frac{R_2}{B} \cdot P_r \right) + \left(\frac{J}{B} \cdot P_t + \frac{R_3}{B} \cdot P_r \right) \right)$$

cost of executing a job on a mobile host =

$$\beta \cdot \left(P_{CPU} \cdot \frac{t_m}{3600} \right)$$

α and β are the weights given to remote and local execution respectively. At the moment $\alpha = \beta = 1.0$.
The job service time and job inter-arrival time of the mobile hosts are obtained from the trace data.
The workload of the fixed hosts are generated using exponential distribution. The fixed hosts are assumed to have an exponential service time of $\mu = 0.1$ and job inter-arrival rate of $\lambda = 1.0$.

We generate the additional workload at the fixed hosts because, in reality, a fixed host may have its own jobs to execute. If so, it can only accept a job transfer request if it has some spare capacity. By taking this approach, we create an environment where a mobile host must compete for the fixed host's resources and so reduce the chances of obtaining over-optimistic results.

IV. Results and Discussion

In the experiments carried out, we want to investigate :

- how the available bandwidth influence job migrations,
- how the mobile host's processor cycle influence the job migration decision,
- the effectiveness of History and ALS compared to LS in extending battery lifetime,
- any other factors which may influence battery lifetime.

We varied the processor cycle time of the mobile relative to the fixed host to see the effect it has on power saving. A processor cycle time of $1/n$ means that the mobile's processor is n times slower than the fixed host.

Simulations were run for groups of 30 users at a time. Detailed figures are given for five of those users, together with an average over all 30 users. In this way, it is possible to see both general behaviour and the way that behaviour may vary between different classes of users. Table 1 gives the characteristic of different classes of users.

A. Bandwidth

The available bandwidth is an important factor in determining if a job can be migrated. We varied the available bandwidth from 9.6 kbps to 100.0 kbps. Figure 1 shows battery lifetime improvement for various bandwidths. The graph shows that for user1, user2 and user3, the duration that the battery lifetime improvement increases as the available bandwidth increases.

For user1, the battery lifetime increased from 2.68 hour (no load sharing) to 3.16 hour (with load sharing) when the available bandwidth is 100.0 kbps. The battery lifetime of user2 increased from 2.52 hour to 3.07 hour. The battery lifetime of user3 increased from 2.57 hour to 3.07 hour. The battery lifetime of user4 and user5 increased from 3.25 hour to 3.27 hour

users	Average job size (bytes)	Average job execution length (sec)	CPU utilisation
User1	8303	10.17	0.69
User2	30441	11.07	0.89
User3	46154	13.50	0.94
User4	22383	0.73	0.01
User5	4980	0.94	0.02

Table 1: Table showing the different characteristics of users.

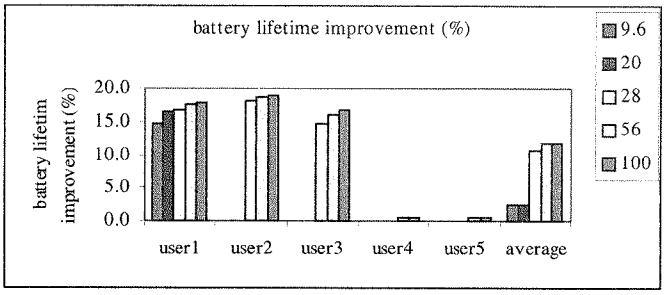

Figure 1: Graph showing the percent of battery lifetime is extended as available bandwidth increases (processor cycle = 1/5).

and from 3.23 hour to 3.27 hour respectively. Table 2 gives a summary of the improvement of battery lifetime of each user. User4 and user5 do not show significant improvement in battery lifetime. The reason why the two users exhibit such behaviour will be discussed in 4.3.

At low bandwidth few jobs were migrated because the amount of time spent transmitting and receiving would consume more power than executing the jobs locally. As the available bandwidth increases, more jobs were migrated (please refer to Figure 2) and therefore, more power saving was achieved.

Note that, even at low bandwidth, user1 transferred many more jobs than the other users. As can be seen from Table 1, user1 execute smaller jobs than the other users. Consequently, it is beneficial for user1 to transfer jobs even at relatively low bandwidth. Figure 3 shows that communication delays decreases as the available bandwidth increases. Note that for user1 and user2, the mean communication delays increases slightly when the available bandwidth increases from 56.0 kbps to 100.0 kbps. The same observation can be made for user3, when the available bandwidth increases from 28.0 kbps to 56.0 kbps. The reason for this observation is that as the available bandwidth increases, jobs which were previously not transferred because it was not feasible to do so, are now selected for transfer. Since these are jobs with relatively large job size, they cause a slight increase in communication delays.

B. Processor Cycle of Mobile Computer

Figure 4 shows that for slower mobiles, more power saving is achieved by migrating jobs. Slower machines require more CPU time to execute a job. The cost of migrating jobs becomes less than the cost of executing them locally. Therefore, on a slow machine, more jobs will be transferred and a significant amount of power can be saved.

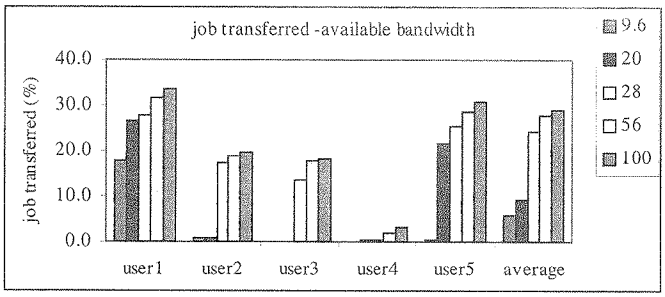

Figure 2: Graph showing percent of job transferred as the available bandwidth increases (processor cycle = 1/5).

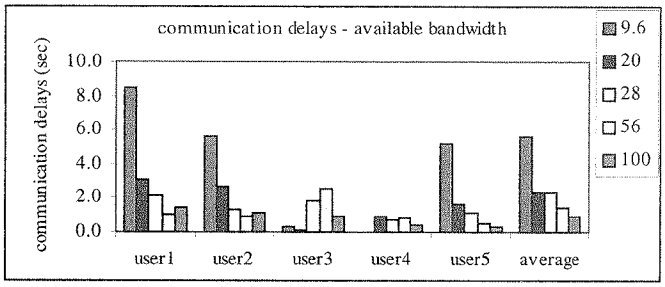

Figure 3: Graph shows that communication delays decreases as the available bandwidth increases (processor cycle = 1/5).

Once again, we can observe that user4 and user5 do not exhibit significant improvement in their battery lifetime.

Apart from extending battery lifetime, migrating jobs gives users access to faster machines. As a result, the mean response time improved. The response time improvement is calculated as follows :

$$\text{RT improvement} = ((RT_a - RT_b)/RT_a)\%$$

where :

RT_a = mean response time without load sharing,

RT_b = mean response time with sharing.

Figure 5 shows that there is a vast improvement in response time as the processor cycle decreases. This can be explained as follows.

The job inter-arrival time is calculated from the trace data. This value remains the same as we decrease the processor cycle. Hence, the jobs still arrive at the same rate even though they are now serviced at a slower rate, causing them to wait longer in queue before being served. When jobs are transferred to a fixed host, the waiting time is reduced drastically. As a result, we see the vast improvement in response time when jobs are transferred from the slow machines.

C. CPU Utilisation

In order to determine why user4 and user5 do not benefit from job migration, we took a closer look at the users' trace data. Upon closer examination, we discovered that the CPU utilisation influences the benefit of job transfers.

We calculated the average CPU utilisation for each trace data. Table 1 shows that the CPU utilisation of user4 and user5

Bandwidth (kbps)	User1		User2		User3		User4		User5		Average for 30 users
	%	hour	%	hour	%	hour	%	hour	%	hour	%
9.6	14.55	0.39	0.00	0.00	0.39	0.01	0.00	0.00	0.00	0.00	2.52
20.0	16.42	0.44	0.00	0.00	1.17	0.030	0.00	0.00	0.62	0.02	2.52
28.0	16.79	0.45	20.24	0.51	17.90	0.46	0.00	0.00	0.62	0.02	10.75
56.0	17.54	0.47	21.03	0.53	18.68	0.48	0.62	0.02	0.62	0.02	11.38
100.0	17.91	0.48	21.83	0.55	19.46	0.50	0.62	0.02	1.24	0.04	11.38

Table 2: A summary of battery lifetime improvement for each user for different bandwidth (processor cycle = 1/5).

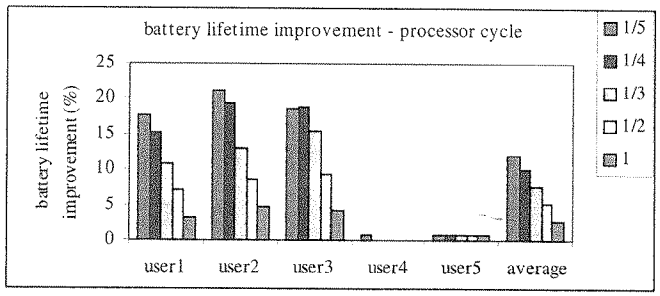

Figure 4: Graph showing that slower mobile computers benefit more from job transfers (bandwidth = 56.0 kbps).

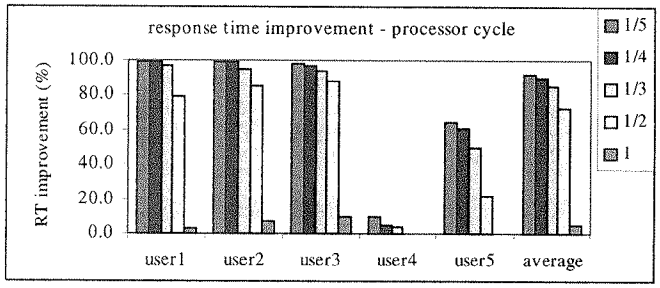

Figure 5: Graph showing RT improvement by giving mobile hosts access to faster machines on the fixed network

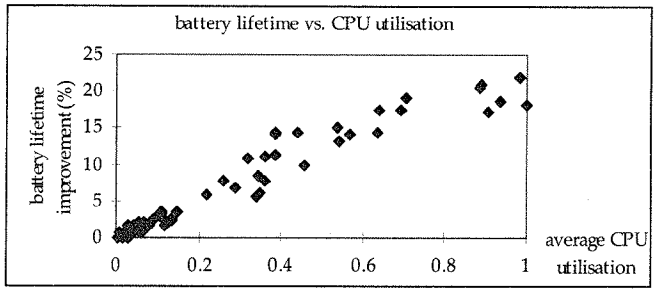

Figure 6: Graph showing the relationship between CPU utilisation and the benefit of job migration. (bandwidth=56.0 kbps; processor cycle=1/5).

are low compared to other users.

$$\text{CPU utilisation} = \frac{\sum J_i}{T}$$

where J_i = CPU time to execute job i (sec)
T = simulation period (sec)

Referring to Figure 2, even though the percentage of jobs transferred for user 5 is comparable to user1, user2 and user3, there is no significant improvement in battery lifetime. This is because since the CPU utilisation is low, transferring jobs does not result in the CPU remaining idle long enough to lead to substantial saving.

The percentage of jobs transferred for user4 is low because most jobs are short-lived and, therefore, not worth migrating. This agrees with the suggestion of [7] that a practical implementation of load sharing should attempt to selects jobs with a relatively high ratio of processing costs to transfer cost.

Figure 6 shows that the benefit of job migration does indeed correspond to average CPU utilisation.

D. History and ALS

Table 3 summarises the simulation results for users using History and ALS compared to LS. From the table, we can see that for all three users, ALS's performance is almost as good as the upper bound LS.

The trace data shows that the execution time of a job is not a constant value. The value may fluctuate. Both History and ALS makes use of an average CPU time in its calculation. Since an average value is used, it is possible that sometimes a incorrect estimate is made regarding power consumption. A wrong prediction may cause either an unnecessary job transfer or cause a job to be executed locally when it should have been migrated. Incorrect predictions result in a waste of battery power.

History makes use of an average value from a previous simulation in its calculation. The same average value for job A is used in the calculation each time job A is executed. Therefore, it is likely that a wrong estimation will be repeated.

On the other hand, ALS which is a dynamic algorithm, uses current information. The average value is updated after each job execution. Consequently, it is capable of adapting its behaviour over time and of improving its estimates. Even though ALS may still make mistakes, these are not as serious as History. Consequently, ALS out performs History. Figure 7 compares the performance of LS, History and ALS.

	battery lifetime extended (%)			job transferred (%)			RT improvement (%)		
	LS	History	ALS	LS	History	ALS	LS	History	ALS
user1	17.54	13.81	16.42	31.38	16.25	30.95	98.73	95.93	98.52
user2	21.03	16.27	20.23	25.95	12.99	25.96	98.96	97.89	98.74
user3	18.68	15.56	17.90	27.42	14.80	27.42	97.85	94.48	97.27

Table 3: Table comparing the performance of LS, History and ALS (Bandwidth = 56.0 kbps; processor cycle = 1/5)

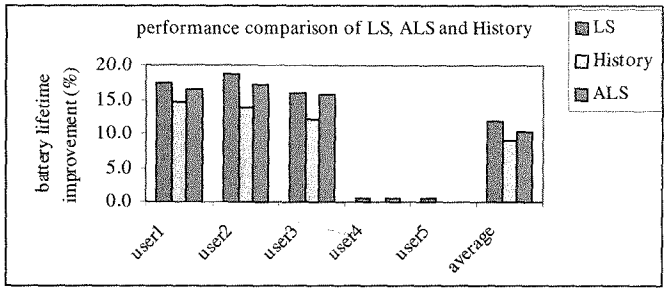

Figure 7: Graphs showing the performance of History and ALS compared to LS

V. Conclusion

We have shown that migrating jobs from a mobile host to a fixed host can extend battery life and improve job response time. The benefits of job migration depend on several factors, i.e. available bandwidth, CPU utilisation and processor cycle of the mobile relative to the fixed host. As the available bandwidth increases, more jobs can be transferred to the fixed host.

We have also established that the benefit of job migration is influenced by CPU utilisation. A mobile host with high CPU utilisation is more likely to benefit from job migrations.

We introduced two dynamic algorithms, i.e. History and ALS. ALS out performs History and its performance is almost as good as the upper bound LS algorithm due to its ability to adapt its behaviour over time.

References

[1] R. Alonso, H.F. Korth, "Database System Issues in Nomadic Computing," *SIGMOD Record*, Vol. 22, Iss. 2, June 1993.

[2] B.R. Badrinath, A. Acharya, T. Imielinski, "Impact of Mobility on Distributed Computing," *Operating Systems Review*, Vol. 27, No. 2, April 1993.

[3] N. Davies, G.S. Blair, A.D. Cross, P.F. Raven, "Mobile Open Systems Technologies for the Utilities Industries," *IEE Colloquium on CSCW Issues for Mobile and Remote Workers*, March, 1993.

[4] F. Douglis, P. Krishnan, B. Marsh, "Thwarting the Power-Hungry Disk," *1994 Winter USENIX Conference*, January 1994.

[5] F. Douglis, R. Caceres, B. Marsh, F. Kaashoek, K. Li, "Storage Alternatives for Mobile Computers," *Proc. of the 1st. Symposium on Operating System Design and Implementation*, USENIX Assc., November 1994.

[6] F. Douglis, P. Krishnan, B. Bershad, "Adaptive Disk Spin-Down Policies for Mobile Computers," *2nd USENIX Symposium on Mobile and Location-Independent Computing*, April 1995.

[7] D.L. Eager, E.D. Lazowska, J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Transactions on Software Engineering*, Vol. 12, No. 5, May 1986.

[8] G.H. Forman, J. Zahorjan, "The Challenges of Mobile Computing," Technical Report, UW CSE #93-11-03, University of Washington, March 1994.

[9] W.C. Grant, "Wireless Coyote : A Computer-Supported Field Trip," *Communications of the ACM*, Vol. 36, No. 5, May 1993.

[10] D.P. Helmbold, D.D.E. Long, B. Sherrod, "A Dynamic Disk Spin-Down Technique for Mobile Computing," *Mobicom* 1996.

[11] C.H. Hsu, J. Liu, "Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems," *Proceedings of the 6th International Conference on Distributed Computing Systems*, May 1986.

[12] T. Imielinski (ed.), H.F. Korth, *Mobile Computing*, Kluwer Academic Publishers, 1996.

[13] Intel Application Note, Pentium(r) Processor (610 75) Power Consumption, Rev. 1.1 , October 1994.

[14] O. Kremian, J. Kramer, "Methodical Analysis of Adaptive Load Sharing Algorithms," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 6, November 1992.

[15] P. Krueger, N.G. Shivarati, "Adaptive Location Policies for Global Scheduling," *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, June 1994.

[16] R. Mirchandaney, D. Towsley, J.A. Stankovic, "Adaptive Load Sharing in Heterogeneous Distributed Systems," *Journal of Parallel and Distributed Computing*, Vol. 9, No. 4, August 1990.

[17] S. Sheng, A. Chandrakasan, R.W. Brodersen, "A Portable Multimedia Terminal," *IEEE Communications Magazine*, December 1992.

[18] K.G. Shin, Y.C. Chang, "Load Sharing in Distributed Real-Time Systems with State-Change Broadcasts," *IEEE Transactions on Computers* , Vol. 38, No. 8, August 1989.

[19] J. Short, R. Bagrodia, L. Kleinrock, "Mobile Wireless Network System Simulation," *Mobicom* '95, November 1995.

[20] A. Svensson, "History, an Intelligent Load Sharing Filter," *Proceedings of the 10th International Conference on Distributed Computing Systems*, 1990.

[21] F. Teraoka, Y. Yokote, M. Tokoro, "A Network Architecture Providing Host Migration Transparency," *ACM SIGCOMM*, September 1991.

[22] Y.T. Wang, R.J.T. Morris, "Load Sharing in Distributed Systems," *IEEE Transactions on Computers*, Vol. C-34, No. 3, March 1985.

[23] T. Watson, "Application Design for Wireless Computing," *IEEE Workshop on Mobile Computing*, December 1994.

[24] M. Weiser, B. Welch, A. Demers, S. Shenker, "Scheduling for Reduced CPU Energy," USENIX 1994 Operating System Design and Implementation Symposium, November 1994.

[25] O. ZeinElDine, M. El-Toweissy, R. Mukkamala, "A Distributed Scheduling algorithm for Heterogeneous Real-Time Systems," Lecture Notes in Computer Science, Vol. 497, 1991.

[26] S. Zhou, "A Trace-Driven Simulation of Study of Dynamic Load Balancing," *IEEE Transactions on Software Engineering*, Vol. 14, No. 9, September 1988.

Biographies

Mazliza Othman is currently reading for a Ph.D. degree at the University College London. She obtained her first degree in Computer Science from Universiti Kebangsaan Malaysia. She obtained her M.Sc. degree in Data Communication Networks and Distributed Systems from University College London. She has worked at Telekom Malaysia as a System Analyst before joining Universiti Malaya as an academic staff.

Stephen Hailes is currently a lecturer in computer science at University College London. He obtained a B.A. in Computer Science from the University of Cambridge in 1987 (M.A. 1991). His Ph.D. work, again at the Computer Laboratory, University of Cambridge, was in the field of Distributed Object-Based Programming languages. Since 1991, he has worked at UCL in the fields of multimedia, networks and distributed systems. After 1995, these interests crystallised around the area of mobile systems, and he now leads the UCL mobile systems group.