Empirical Evaluation of Dynamic Local Adaptation for Distributed Mobile Applications

Pablo Rossi and Caspar Ryan

School of Computer Science & IT, RMIT University
Melbourne, Victoria, Australia
{pablo,caspar}@cs.rmit.edu.au

Abstract. Distributed mobile applications operate on devices with diverse capabilities, in heterogeneous environments, where parameters such as processor, memory and network utilisation, are constantly changing. In order to maintain efficiency in terms of performance and resource utilisation, such applications should be able to adapt to their environment. Therefore, this paper proposes and empirically evaluates a local adaptation strategy for mobile applications, with 'local' referring to a strategy that operates independently on each node in the distributed application. The strategy is based upon a series of formal adaptation models and a suite of mobile application metrics introduced by the authors in a recent paper. The experiments demonstrate the potential practical application of the local adaptation strategy using a number of distinct scenarios involving runtime changes in processor, memory and network utilisation. In order to maintain application efficiency in response to these changing operating conditions, the system reacts by rearranging the object topology of the application by dynamically moving objects between nodes.

1. Rationale

It has been recognised that applications with distributed components differ from their traditional non-distributed counterparts along a number of important dimensions including communication type, latency, concurrency, partial versus total failure, and referencing/parameter-passing strategies [1].

Rapid advances in hardware technology have given current laptop machines the processing power of servers only a few years old, with smaller mobile devices such as Intel XScale based PDA's now having CPU's running at hundreds of megahertz with 64MB or more of program memory. Phone technology is also rapidly advancing with current generation phones able to run application code using standardised platforms such as Java 2 Micro Edition (J2ME) [2], Symbian OS [3], and Microsoft .NET Compact Framework [4]. As such, an increasingly diverse range of potential devices, including desktop PCs, laptops, PDAs and smartphones, are capable of running object oriented application code in a virtual machine environment, and thus actively participate as part of a distributed mobile application.

With such a diversity of possible devices, previous challenges such as communication latency, failure mode and concurrency are now complemented by the issue of maintaining efficiency and quality of service in a significantly more heterogeneous networking and execution space.

Although these challenges appear daunting, Ryan and Perry [5] demonstrated in an empirical study that there are substantial benefits to be realised by end-users and application service providers, through better utilisation of the computing power of client side devices. The primary caveat to such an approach is minimising the additional developer effort required to produce applications with fatter or adaptive smart clients that can take advantage of increasing client-side computing resources.

In order to address the challenge of writing distributed applications for heterogeneous environments whilst minimising the amount of additional developer effort, a context aware adaptive mobile application framework called MobJeX [6] is being developed, in conjunction with the work described in this paper, as part of the Applications Program of the Australian Telecommunications Cooperative Research Centre (ATcrc). MobJeX is a software development platform with a middleware component enabling dynamic application adaptation based on object mobility in response to environmental changes detected by runtime resource monitoring.

In this context, application adaptation refers to the ability of an application, or the underlying middleware, to modify its behaviour in response to changes in environmental context, (e.g. available network bandwidth or CPU load). In the case of MobJeX, adaptation is achieved via object mobility, in which individual system components, potentially down to the discrete object level, can migrate through the system whilst maintaining location transparency via remote object references.

Given the challenge of writing applications that can adapt to run more efficiently in diverse operating environments, and given the existence of a prototypical application framework that includes fully functional system monitoring and transparent object mobility, this paper is concerned with deriving, implementing and testing a local adaptation algorithm using a number of realistic scenarios applied to an actual distributed application running within the MobJeX framework. The concept of local adaptation refers to a strategy whereby the algorithm operates independently at each system node in order to coordinate the adaptation of the distributed application.

The rest of this paper is organised as follows: Section 2 briefly summarises a suite of metrics and models for mobile applications developed by the authors in recent work. This serves as a formal basis for the derivation of the local adaptation algorithm presented in this paper. Section 3 provides a literature review of existing work in application adaptation of distributed systems, particularly those featuring object mobility as a means of adaptation. Section 4 details the derivation and operation of the new adaptation algorithm itself, while section 5 describes a series of empirical studies using a real application running on the MobJeX framework in order to demonstrate the potential practical application of the algorithm using a number of common scenarios. Section 6 finishes with a summary, conclusions and a discussion of opportunities for future work.

2. Background

In previous work [7], the present authors proposed, and mathematically modelled using metrics, a number of software and efficiency attributes likely to have an impact on the execution of mobile applications. This suite of metrics and their representative models were identified from a critical analysis of the problem domain after a review of the metrics literature, and empirically validated through the formulation of concrete hypotheses expressing the intuitive relationships between the software and efficiency attributes. Table 6 in the Appendix lists this set of attributes and associated metrics, and provides a summary of the relationships among them.

The authors also identified a practical application of the metric based models, involving runtime application adaptation; particularly in terms of runtime object topology, in which the clustering of application objects and placement of object clusters to nodes varies in response to changing environmental conditions. Two decision-making strategies were proposed. Firstly, a global adaptation strategy whereby the application is analysed as a whole, with optimisation performed in terms of mapping object clusters to nodes. Secondly, a more dynamic and reactive technique called local adaptation involves individual nodes moving one object at a time to other hosts when either a performance or resource utilisation threshold is met.

In this previous study a preliminary adaptation algorithm was presented in order to illustrate the practical application of the metrics and models within an existing mobile application framework called MobJeX [6]. This paper extends that work by deriving and implementing a new local adaptation algorithm that provides a substantial amount of flexibility in terms of preference and weighting given to individual quality related attributes. This algorithm, is described in detail in section 4, and evaluated empirically in section 5 using a number of live scenarios involving different adaptation policies applied to a real mobile application running on the MobJeX framework.

Before describing the new algorithm it is first appropriate to review existing work on application adaptation, particularly that involving object mobility. Therefore, the following section examines a number of previous studies, identifying aspects that have been incorporated into the work described in this paper, as well as highlighting limitations and thus providing further rationale for the algorithm presented in section 4.

3. Literature Review – Application Adaptation

Adaptation for distributed applications has received significant research attention in the last few years. Earlier approaches focused on adaptation for client-server applications, [8-11], while more recent efforts address adaptation for parallel applications [12] or generic distributed applications where individual components reside on peers [13-20].

The common feature of these papers is the monitoring of the execution environment to detect resource utilisation and capacity changes and trigger an adaptation mechanism to improve performance. The most commonly monitored resource attributes, which are measured either in terms of total capacity, utilisation or both, are (in order of frequency): network, memory, processor and battery. In addition, some of the studies consider the type of information exchange between [10, 11], or the nature of interaction among, distributed application components [12, 13, 18].

Note however that none of the existing approaches consider the impact of adaptation on resource utilisation and performance at the same time, as is done by the adaptation algorithm described in section 4 and evaluated in section 5 of this paper. Furthermore, none of these proposals take into account software attributes of application components nor do they make decisions based on a formal model using empirically validated metrics, again as is the case of the strategy presented herein.

The actual mechanisms employed to achieve adaptation include object migration [12, 13], the selection of alternative methods [16, 19], the substitution of object implementations [8, 17, 21], and middleware reconfiguration [9, 14, 15, 20]. In some cases the information exchanged by components is adapted, rather than the components themselves [10, 11]. How this is done can depend on factors such as whether the information is binary or text-based. Another adaptation strategy involves using location information to place servers close to the clients [18].

The approaches that employ object migration as the adaptation mechanism [12, 13], are the most relevant to the adaptation algorithm presented in this paper. However, in contrast to this new strategy, neither of the existing approaches is transparent from the perspective of the application developer, the significance of which is described in the following paragraph. In fact with the exception of [10, 14], which are not concerned with adaptation via object migration, none of the existing adaptation approaches are application transparent.

Jing [22] identifies three broad classes of application adaptation. First is *laissez-faire adaptation,* a strategy in which the application is entirely responsible for triggering and implementing adaptation. Second is *application aware adaptation*, wherein applications explicitly interact with middleware services to facilitate adaptation. Last is *application transparent adaptation,* which is the most desirable but most difficult to achieve in practice. In this case, both the decision to trigger adaptation and the strategy for executing it are controlled independently of the application via middleware. This type of adaptation, which is the subject of this paper, is the most appealing from the perspective of the software developer, since the software can be implemented using conventional techniques while still realising the potential benefits of adaptation.

Another point of distinction between the reviewed studies is that there is no clear tendency with regards to the scope of adaptation. Some approaches adapt the application as a whole [9-11, 14-16], while others include the ability to adapt discrete application components [8, 12, 13, 17-21]. The latter approach is more flexible since it offers more adaptation options and finer granularity, but is more complex since it requires a more sophisticated implementation mechanism.

Finally, none of the previous approaches, with one exception [12], distinguish between local (or decentralised) and global (or centralised) adaptation as discussed in the previous section. Moreover, in contrast to this paper, the study by Garti et al. [12] is concerned with multi-threaded applications and tries to achieve better performance by distributing application threads to different machines and executing them concurrently.

4. Local Adaptation Strategy

This section describes the main contribution of this paper, which is a local adaptation algorithm that optimises runtime object topology, through the clustering of application objects and placement of object clusters to nodes, in response to changing environmental conditions.

```
do {
    maxScore = 0.5
    maxObject = null, maxNode = null
    for each mobile object o in local node do
       for each remote node n do
           score = evaluate(o, n)
           if (score > maxScore) then
               maxScore = score
               maxObject = o
               maxNode = n
           end if
        end for
    end for
    if (maxScore > 0.5) then
        move maxObject to maxNode
   end if
while (maxScore > 0.5)
```

Fig. 1. Local Adaptation Algorithm, Basic Flow of Control

4.1. Basic Algorithm

At the abstract level, the local adaptation algorithm operates according to Fig. 1 in which individual nodes move objects to other hosts when criteria related to efficiency (performance versus resource utilisation) [23] are met. Although the basic working loop is similar to that presented in [7], the scoring function `evaluate()`, is significantly more capable in terms of the following: Firstly, this algorithm allows multiple efficiency sub-attributes and their inter-relationships to be considered in a single pass. Secondly, the algorithm allows specific sub-attributes to be prioritised through weighting (e.g. response time and thus performance could be favoured over network utilisation). Finally, the algorithm allows the specification of the extent to which a certain attribute should be favoured over others. For example, should performance be increased by a small amount given a high cost in resource utilisation?

The algorithm evaluates, using metric-based models [7], possible migration options based on the available local mobile objects and remote nodes. This can be used for ranking purposes, or for selecting the highest score provided it is greater than a predetermined threshold, in order to establish which object migration to carry out. The algorithm stops when the highest score no longer exceeds the threshold or when there are no more local mobile objects.

An explanation of how the sub-algorithm ‘evaluate’ produces its scores is given in the following section.

4.2. Using Metrics to Calculate Decision Making Scores

There are a number of possible general approaches to decision making based on multiple attributes, which differ in terms of how they specify criteria for the decision making process. These include linear [24] and non-linear [25] approaches which can be specified for the general case (independent of efficiency) according to equations 1 and 2 respectively.

$$S = (W_1 I_1 + W_2 I_2 + ... + W_m I_m) \quad \mathbf{(1)}$$

In order to evaluate such functions, and thus produce a decision making score (S) that can be used to rank and execute actions, the level of satisfaction of the individual indicators (I_i) must be calculated. This is done by normalising the values to the unitary interval ($0 \leq I_i \leq 1$) where: 0.5 means the indicator just equals its satisfaction criterion; > 0.5 means that as the value of the indicator increases towards 1, the greater it satisfies the individual criterion up to the maximum level of satisfaction of 1, corresponding to the maximum measurable value for the metric associated with the indicator. Conversely, < 0.5 means that as the indicator value decreases towards 0, the less it satisfies the criterion down to the minimum level of satisfaction of 0, corresponding to the minimum possible value for the associated metric.

Furthermore, both the linear and non linear variations of the aggregate decision making function include weights (W_i), to represent the relative importance of the individual indicators when calculating the decision making score S. A further requirement of both functions is that ($W_1 + W_2 + ... + W_m$) = 1, where $W_i \geq 0$ for $i = 1 ... m$.

Equation 2 is a non-linear 'weighted power mean' [25], which in addition to allowing the specification of relative importance via weights, also allows the specification of whether an indicator is mandatory, alternative, or neutral.

$$S(r) = (W_1 I_1^r + W_2 I_2^r + \ldots + W_m I_m^r)^{1/r} \quad \mathbf{(2)}$$

where $-\infty \leq r \leq +\infty$, $S(-\infty) = \min(I_1, I_2, ..., I_m)$ and $S(+\infty) = \max(I_1, I_2, ..., I_m)$.

The power r is a real number parameter selected to achieve the desired indicator relationship of the aggregation function. Equation 2 is equivalent to equation 1 when $r = 1$, which models the neutrality relationship where all indicators are considered equally with significance attributed only to their value and weighting. Equation 2 is supra-additive for $r > 1$, which models indicator replaceability (or disjunction) meaning that one or more higher indicators are favoured at the cost of lower indicators. Alternatively, it is sub-additive for $r < 1$ (with $r \neq 0$), thereby modelling indicator simultaneity (or conjunction), thus favouring the situation where there are no low indicators.

For example, consider the following situation where $S = (0.5\, I_1^r + 0.5\, I_2^r)^{1/r}$. Table 1 shows the effect of r on S for different values of I_1 and I_2. Firstly, when $r = 1$, S is the average of I_1 and I_2. Secondly, when $r > 1$, S is greater than the average (i.e. closer to the highest indicator I_2). Finally, when $r < 1$, S is less than the average (i.e. closer to the lowest indicator I_1).

Table 1. Effect of the parameter *r* on the scoring model *S*

r	I_1	I_2	*S*	I_1	I_2	*S*	I_1	I_2	*S*
10	0.5	0.5	0.5	0.4	0.6	0.56	0.1	0.9	0.84
1	0.5	0.5	0.5	0.4	0.6	0.50	0.1	0.9	0.50
-10	0.5	0.5	0.5	0.4	0.6	0.43	0.1	0.9	0.11

The ‘weighted power mean’ approach can be specialised with the efficiency sub-attributes (indicators) resource utilisation and response time, to model decision making and thus facilitate adaptation based on efficiency (Equations 3 and 4). These specific equations serve as the computational basis for the sub-algorithm ‘evaluate’ in Fig. 1 and the empirical studies of the adaptation behaviour of the local adaptation algorithm, which are presented in the following section. It should be noted that similar equations could be specified for other quality attributes (and their sub-attributes) as has been done in the case of web applications [26]. This is however beyond the scope of this paper and thus quality attributes such as reliability, which could be relevant to local adaptation, are left as the subject of future work.

$$S_E = (W_{MU} I_{MU}^r + W_{NU} I_{NU}^r + W_{PU} I_{PU}^r + W_{RT} I_{RT}^r)^{1/r} \tag{3}$$

$$I_i = 0.5 + 0.5\,(d - k) / (2 \times max) \tag{4}$$

where $d = ru_d$ or rt_d, $k = ru_k$ or rt_k (see Equations 5 and 6 in the Appendix), and $max = ru_{max}$ or rt_{max}.

5. Empirical Evaluation

A series of empirical studies were conducted by deploying a prototype of a Taxi Dispatching System (TDS) on the MobJeX framework [6], in order to evaluate the adaptation algorithm presented in the previous section. The TDS application was chosen because it is simple enough to be described within this paper, but complex enough in terms of its design, functionality and object topology, to provide meaningful evaluation of the metric-based adaptation strategy presented in the previous section. Furthermore, the TDS application has sufficient scope to suggest explicit directions for future work. Note that the TDS application design is based on five main objects (a location manager [lm], client manager [cm], job manager [jm], taxi manager [tm], and user interface [ui]). A description of TDS in terms of its functional requirements and main use cases appears in [7].

Software metrics were collected offline via a static analysis of the TDS source code, whereas resource utilisation and performance metrics were obtained online during execution via the resource monitor component of MobJeX. Furthermore, the actual adaptation decisions produced by the local adaptation algorithm of section 4 were carried out using MobJeX to transparently (i.e. without impacting application state) migrate objects between nodes at runtime. The experimentation was conducted in a research laboratory with a 100 Mbps Ethernet network, which was isolated from the rest of the university to eliminate the confounding effect of external traffic. Furthermore, since the TDS application was relatively small with four main mobile objects[1] (and a non-mobile object: [ui]), we used a small machine cluster with three identical nodes (1 GHz, 512MB, Windows XP) and used software to cap the network bandwidth at 11 Mbps to emulate a slower 802.11b wireless network.

Note that it is not the intention of this paper to test the efficiency or accuracy of the metrics collection process itself but rather the effectiveness of the local adaptation algorithm when fed appropriate metrics. It is the subject of future work to look at the impact of the actual metrics collection process and how this must be factored into the decision making process when deciding if a given object topology is more efficient than another. Furthermore, a working implementation of this strategy will require the existence of some protocol among the nodes to guarantee the execution of the adaptation at only one node at any given time. This together with careful selection of parameters will suffice in most cases to prevent undesirable side effects such as system thrashing and 'pinball' migration (i.e. an object keeps migrating between two or more nodes).

Three separate experiments, each testing a number of variations of the tuning parameters, were conducted in order to test the main characteristics of the local adaptation algorithm. These experiments involved adaptation in response to changes in processor, memory and network utilisation respectively, with varying prioritisation of performance versus resource utilisation. In addition, different tuning parameters for the algorithm, in terms of attribute weights and values of 'r' (see equation 2 in section 4) were chosen in order test the impact of parameter choices.

For each of the experiments the following efficiency metrics were collected: 1) Performance in terms of average scenario response time in milliseconds; 2) The standard deviation of processor utilisation across nodes 3) The standard deviation of memory utilisation across nodes 4) The standard deviation of network utilisation across nodes. Note that the standard deviation measurements of 2-4 served as an indication of resource utilisation in terms of load balance, where a lower standard deviation represented a more even balance across nodes. Additionally, all the resource utilisation measurements were presented as percentages, derived from the ratio usage/capacity of resources, for a single node (see Table 6 in the appendix).

These four measurements serve as dependent variables for the experiments and were collected at three different stages: 1) Before a change in resource utilisation has occurred (Initial State); 2) After an event has been triggered to signify a change in the utilisation of a resource, where no adaptation has been performed (No-Adaptation Final State); and 3) After the same event, but where the local adaptation algorithm has been executed to optimise the object topology of the application in response to the change in resource utilisation (Adaptation Final State). This data allows us to directly measure the effectiveness of the local adaptation algorithm in terms of maintaining efficiency in a dynamically changing environment.

5.1. Adapting to Changes in Processor Utilisation

The experiment in this section evaluates the impact of the adaptation strategy on efficiency when processor utilisation changes. This was done using the following

[1] These main objects were effectively object clusters since they held references to a number of smaller worker objects that were also moved as part of the migration process of MobJeX

linear ($r = 1$) scoring model that applies equal weights to performance and processor utilisation, whilst ignoring the other indicators:

$S = (0\ I_{MU} + 0\ I_{NU} + 0.5\ I_{PU} + 0.5\ I_{RT})$, with $r = 1$.

Measurements were taken based on the following three states, with thresholds rt_k = 1ms and ru_k = 1% used to invoke the maximum adaptation outcome.

- *Initial State (IS)*: TDS executing centrally using only node [Z].
- *Event*: The processor load in node [Z] increases significantly (up to 90% utilisation).
- *Adaptation Final State (AFS)*: TDS executing in a distributed manner using nodes [X], [Y] and [Z].
- *No-Adaptation Final State (NFS)*: the same as IS.

Expected outcome. After the event where processor utilisation of node [Z] increases to 90% utilisation, the adapted application should: 1) Perform better than the non-adapted state due to the extra processor time available, and 2) Processor usage should be more balanced, thus resulting in improved overall efficiency.

Algorithm Trace. Fig. 2(a) shows the complete algorithm trace for model *S* and Fig. 2(b) illustrates the adaptation (migration) process schematically.

In the first decision, [tm] and [lm] get high I_{RT} because their methods are characterised by a high Number of Executed Instructions (*NEI*, see Appendix) and a low Size of Serialised Parameters (*SSP*). This implies a low invocation time (*IT*) and thus the overhead of a remote invocation becomes less significant since the Execution Time (*ET*), which does not change significantly upon migration, is the main contributor to overall Response Time (*RT*) since $RT = ET + IT$. However, [tm] gets a higher I_{PU} than [lm], and hence a higher decision score *S*, due to the higher Number of Invocations (*NI*) of its methods.

At the time of the second decision, following the first migration, node [Z] is less loaded. Therefore, moving further objects affects performance negatively ($\downarrow I_{RT}$). Consequently, objects such as [cm] with high NEI but low NI score higher than objects such as [lm] since they do not significantly affect performance but do improve processor load balance ($\uparrow I_{PU}$). [cm] is not coupled to [tm], so locating [cm] in a different node ([Y]) does not add remote invocations (which would affect performance) but does improve load distribution.

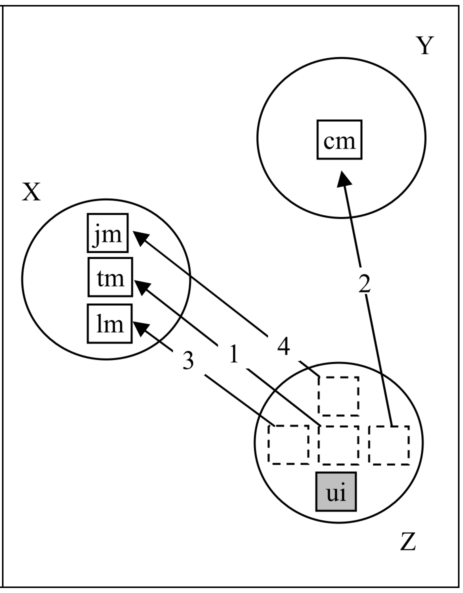
In the third decision, moving [lm] to node [X] does not negatively affect performance due to remote invocations because [lm] is only coupled to [tm], which resides in node [X], thus slightly improving processor load distribution.

Finally, in the last decision, moving [jm] to node [X] leaves performance at the same level, since it is mostly coupled to [tm], but slightly improves processor utilisation, by scoring just above the decision making threshold of 1%.

Results. Table 2 shows the efficiency metrics for the 3 states described in this section. As expected, after the environmental change and subsequent adaptation, the execution of the adapted application performs considerably better than the non-adapted application and the processor usage is more balanced, thus resulting in improved overall efficiency.

```
M. [tm] to [X] => S: 0.6322
M. [tm] to [Y] => S: 0.6322
M. [lm] to [X] => S: 0.5599
M. [lm] to [Y] => S: 0.5599
M. [jm] to [X] => S: 0.4491
M. [jm] to [Y] => S: 0.4491
M. [cm] to [X] => S: 0.5146
M. [cm] to [Y] => S: 0.5146
>> Migrate [tm] to [X]
M. [lm] to [X] => S: 0.5058
M. [lm] to [Y] => S: 0.5016
M. [jm] to [X] => S: 0.4848
M. [jm] to [Y] => S: 0.4770
M. [cm] to [X] => S: 0.4930
M. [cm] to [Y] => S: 0.5061
>> Migrate [cm] to [Y]
M. [lm] to [X] => S: 0.5052
M. [lm] to [Y] => S: 0.4999
M. [jm] to [X] => S: 0.5015
M. [jm] to [Y] => S: 0.4819
>> Migrate [lm] to [X]
M. [jm] to [X] => S: 0.5066
M. [jm] to [Y] => S: 0.4732
>> Migrate [jm] to [X]
```

(a)



(b)

Fig. 2. Algorithm trace and the corresponding schematic diagram for scoring model *S*

Table 2. Efficiency metrics for scoring model *S*

	IS	AFS	NFS
Avg. SRT	13.78 ms	24.76 ms	40.09 ms
Std. Dev. PU	37.70 %	43.45 %	51.61 %

5.2. Adapting to Changes in Memory Utilisation

The experiments in this section evaluate the impact of the adaptation strategy on efficiency attributes, when memory utilisation changes. Therefore, in the scoring models of this section different weights were assigned to the memory utilisation and response time indicators, while the weight of network and processor utilisation remained fixed at zero. The algorithm was tested with various tuning parameters in terms of indicator weights and values of the parameter *r* as described by the following five variations: 1) Neutral model with equal weights; 2) Neutral model with performance favoured over memory utilisation; 3) Neutral model with memory utilisation favoured over performance; 4) Disjunctive model; and 5) Conjunctive model. The following memory utilisation event and three states apply to all five variations of the scoring function.

- *Initial State (IS)*: TDS executing centrally using only node (Z).
- *Event*: The amount of free memory in node [Z] decreases significantly (by 256 MB).

- *Adaptation Final State (AFS)*: TDS executing in a distributed manner using nodes [X] and [Z].
- *No-Adaptation Final State (NFS)*: the same as IS.

In all cases the thresholds $rt_k = 1$ ms and $ru_k = 1\%$ were used to invoke the maximum adaptation outcome.

Neutral model with equal weights. $S_1 = (0.5\ I_{MU} + 0\ I_{NU} + 0\ I_{PU} + 0.5\ I_{RT})$, $r = 1$. This scoring model tests how the algorithm reacts when the same weight is assigned to performance and memory utilisation, and the other indicators are ignored (value of 0), for the linear case of $r = 1$.

Expected outcome. After the event where memory utilisation of node [Z] increases by 256MB, the adapted application should perform better than the non-adapted state due to less paging activity, and the memory usage should be more balanced, thus resulting in improved overall efficiency.

Algorithm Trace. Fig. 3 shows a partial algorithm trace for model S_1. Note that for brevity, the final stages of the trace, where no migration decisions occur ($S_1 < 0.5$), are omitted. The fact that [cm] gets the highest score can be explained in two parts. Firstly, [cm] is characterised by the highest Object Memory Size (*OMS,* see Appendix), therefore, the decision to move this object implies a larger mu_d, which in turn implies a larger I_{mu} (see equation 4). Furthermore, the methods of [cm] are characterised by a low Number of Invocations (*NI*) and thus moving this object does not imply a low rt_d, thereby avoiding a low I_{rt}. Note that the performance indicator I_{rt} is considered in the context of overall Scenario Response Times (*SRT*), following a chain of method calls, rather than the individual response time of a single method. Consequently, the low number of invocations has a minimal effect on the overall performance of the application scenario to which the method calls belong. Finally, although another object [lm], scores slightly better for I_{rt}, it does not balance the load as much as [cm], therefore scoring a lower I_{mu} and thus a lower overall score S_1.

```
Move [tm] to [X] => S: 0.4938
Move [tm] to [Y] => S: 0.4938
Move [lm] to [X] => S: 0.5248
Move [lm] to [Y] => S: 0.5248
Move [jm] to [X] => S: 0.3388
Move [jm] to [Y] => S: 0.3388
Move [cm] to [X] => S: 0.5266
Move [cm] to [Y] => S: 0.5266
>> Migrate [cm] to [X]
...
>> No Migration
```

Fig. 3. Algorithm trace for model S_1

	IS	AFS	NFS
Avg. SRT	19.57 ms	26.97 ms	43.09 ms
Std. Dev. MU	9.14 %	34.12 %	37.54 %

Table 3. Efficiency metrics (scoring model S_1)

Results. Table 3 shows the efficiency metrics for the 3 states described above. As expected, after the environmental change and subsequent adaptation, the execution of the adapted application performs considerably better than the non-adapted application and the memory usage is marginally more balanced, hence resulting in improved overall efficiency.

Neutral model with performance outweighing memory utilisation. $S_2 = (0.4\ I_{MU} + 0\ I_{NU} + 0\ I_{PU} + 0.6\ I_{RT})$, $r = 1$. This scoring model tests how the algorithm reacts when different weights are assigned to the indicators of interest, with performance considered more important than memory utilisation, again for the neutral (linear) case where $r = 1$.

Expected outcome. This should be similar to the previous case; however performance should be higher than S_1, possibly at the expense of memory load balance across nodes.

Algorithm Trace. Fig. 4. shows the partial algorithm trace for model S_2. Since performance is more important (i.e. has a higher weight) than memory usage, objects with a high performance indicator I_{RT} score higher overall for S_2 because of the additional weighting compared with I_{MU}. In this case, [lm] gets the highest I_{RT} because although its methods are characterised by a high Number of Executed Instructions (*NEI*), the low Size of Serialised Parameters (*SSP*) implies a lower invocation time (*IT*) and thus the overhead of a remote invocation becomes less significant since the Execution Time (*ET*), which does not change upon migration, is the main contributor to overall Response Time (*RT*) since $RT = ET + IT$.

Results. Table 4 shows the efficiency metrics for the 3 application states. Again, overall efficiency is improved in the adapted versus non-adapted state, however as predicted, the performance gain is greater and the memory load less balanced than model S_1.

```
M. [tm] to [X] => S: 0.4895
M. [tm] to [Y] => S: 0.4895
M. [lm] to [X] => S: 0.5290
M. [lm] to [Y] => S: 0.5290
M. [jm] to [X] => S: 0.3071
M. [jm] to [Y] => S: 0.3071
M. [cm] to [X] => S: 0.5288
M. [cm] to [Y] => S: 0.5288
>> Migrate [lm] to [X]
...
>> No Migration
```

Fig. 4. Algorithm trace for scoring model S_2

```
M. [tm] to [X] => S: 0.5152
M. [tm] to [Y] => S: 0.5152
M. [lm] to [X] => S: 0.5040
M. [lm] to [Y] => S: 0.5040
M. [jm] to [X] => S: 0.4974
M. [jm] to [Y] => S: 0.4974
M. [cm] to [X] => S: 0.5154
M. [cm] to [Y] => S: 0.5154
>> Migrate [cm] to [X]
...
>> No Migration
```

Fig. 5. Algorithm trace for scoring model S_3

Table 4. Efficiency metrics (scoring model S_2)

	IS	AFS	NFS
Avg. SRT	19.57 ms	23.90 ms	43.09 ms
Std. Dev. MU	9.14 %	36.10 %	37.54 %

Neutral model with memory utilisation outweighing performance. $S_3 = (1\ I_{MU} + 0\ I_{NU} + 0\ I_{PU} + 0\ I_{RT})$, $r = 1$. This scoring model is similar to S_2, however in this case memory utilisation is completely favoured over performance, with a weight of one and zero respectively.

Expected Outcome. After the event, the memory utilisation of the adapted execution should be more balanced than the non-adapted execution, possibly at the expense of performance.

Algorithm Trace. The algorithm trace for model S_3 can be seen in Fig. 5. Since memory utilisation is more important than performance, objects with a high I_{MU} score a greater value for S_3 than objects such as [lm] which have a high I_{RT} but lower I_{MU}. [cm] gets the highest I_{MU} for the same reason as S_1. However, although the indicator values change, the adaptation decision remains the same as S_1.

Results. The results are not repeated since the migration decision, and thus the results, are the same as S_1 (Table 3). This occurred because although the different weighting affected the indicator scores, the size of the objects compared with the memory capacity (MC) of the nodes, and the size of the memory utilisation event (256MB), was not significant enough to exceed the threshold of 1% after the initial case of moving [cm].

Disjunctive Model. $S_4 = (0.5\ I_{MU}^{\ r} + 0\ I_{NU}^{\ r} + 0\ I_{PU}^{\ r} + 0.5\ I_{RT}^{\ r})^{1/r}$, $r = 8$.

Expected outcome. Although the indicators of performance and memory usage are equally important, setting $r > 1$ means that a big improvement on one attribute, even at the cost of deterioration of another, is preferred over a small or medium improvement on both attributes. Therefore, after the event, adapted execution should perform either significantly better than the non-adapted execution or the memory usage should be considerably more balanced.

Algorithm Trace. The algorithm trace for model S_4 can be seen in Fig. 6. The decision to move [lm] achieves the highest value for a relevant indicator, and thus its associated decision score S_4 is also higher.

```
M. [tm] to [X] => S: 0.4970
M. [tm] to [Y] => S: 0.4970
M. [lm] to [X] => S: 0.5276
M. [lm] to [Y] => S: 0.5276
M. [jm] to [X] => S: 0.4562
M. [jm] to [Y] => S: 0.4562
M. [cm] to [X] => S: 0.5274
M. [cm] to [Y] => S: 0.5274
>> Migrate [lm] to [X]
...
>> No Migration
```

Fig. 6. Algorithm trace for scoring model S_4

```
M. [tm] to [X] => S: 0.4757
M. [tm] to [Y] => S: 0.4757
M. [lm] to [X] => S: 0.5075
M. [lm] to [Y] => S: 0.5075
M. [jm] to [X] => S: 0.1814
M. [jm] to [Y] => S: 0.1814
M. [cm] to [X] => S: 0.5189
M. [cm] to [Y] => S: 0.5189
>> Migrate [cm] to [X]
...
>> No Migration
```

Fig. 7. Algorithm trace for scoring model S_5

Results. Again, the results are not repeated since the migration decision, is the same as S_2 (Table 4). This is in line with the expected outcome since the highest score is produced by the migration option with the highest indicator (I_{RT}).

Conjunctive model. $S_5 = (0.5\, I_{MU}^r + 0\, I_{NU}^r + 0\, I_{PU}^r + 0.5\, I_{RT}^r)^{1/r}$, $r = -100$.

Expected outcome. Although the indicators of performance and memory usage are equally important, setting $r < 1$ produces a lower score if any of the indicators is low regardless of whether any of the other attributes have a high value. Hence, a small or medium improvement on both attributes is preferred over a big improvement on one at the cost of deterioration of another. Therefore, the adapted application should be more efficient overall than the non-adapted application.

Algorithm Trace. Fig. 7 illustrates the algorithm trace for model S_5. The decision to move [cm] achieves the highest minimum value for a relevant indicator, and thus its associated decision score S_5 is also higher.

Results. Here the value of r = -100 was chosen since it demonstrated that even an extremely low value of r did not change the result from the default case of $r = 1$, since this case had already chosen the migration option that produced the result with the highest minimum indicator. As such, the scenario results, which were the same as the linear case for S_1 in Table 3, were according to expectation.

5.3. Adapting to Changes in Network Utilisation

This final scenario assigns different weights to the network utilisation and response time indicators, while fixing the weights of memory and processor utilisation at zero, in order to test the ability of the local adaptation algorithm to respond to changes in network utilisation. As in the previous sub-section, the algorithm was tested with various tuning parameters in terms of indicator weights and values of the parameter r as follows:

- $S'_1 = (0.5\, I_{RT}^r + 0\, I_{PU}^r + 0\, I_{MU}^r + 0.5\, I_{NU}^r)^{1/r}$, $r = 1$
- $S'_2 = (0.1\, I_{RT}^r + 0\, I_{PU}^r + 0\, I_{MU}^r + 0.9\, I_{NU}^r)^{1/r}$, $r = 1$
- $S'_3 = (0.9\, I_{RT}^r + 0\, I_{PU}^r + 0\, I_{MU}^r + 0.1\, I_{NU}^r)^{1/r}$, $r = 1$
- $S'_4 = (0.5\, I_{RT}^r + 0\, I_{PU}^r + 0\, I_{MU}^r + 0.5\, I_{NU}^r)^{1/r}$, $r = 100$
- $S'_5 = (0.5\, I_{RT}^r + 0\, I_{PU}^r + 0\, I_{MU}^r + 0.5\, I_{NU}^r)^{1/r}$, $r = -100$

As in the previous cases the thresholds $rt_k = 1$ ms and $ru_k = 1\%$ were used to invoke the maximum adaptation outcome. The following three states apply to all five variations based on a network utilisation event:

- *Initial State (IS)*: TDS executing in a distributed manner using nodes [X] and [Z] as in AFS of the previous section.
- *Event*: Increased network utilisation causes the network bandwidth available to node [X] to decrease significantly to 0.55 Mbps.

- *Adaptation Final State (AFS)*: TDS executing in a distributed manner using nodes [Y] and [Z].
- *No-Adaptation Final State (NFS)*: the same as IS.

Table 5. Efficiency metrics for model S'_1

	IS	AFS	NFS
Avg. SRT	39.34 ms	38.09 ms	43.35 ms
Std. Dev. NU	1.86 %	3.12 %	36.90 %

```
M. [cm] to [Y] => S: 0.5786
M. [cm] to [Z] => S: 0.5468
>> Migrate [cm] to [Y]
```

Fig. 8. Algorithm trace for model S'_1

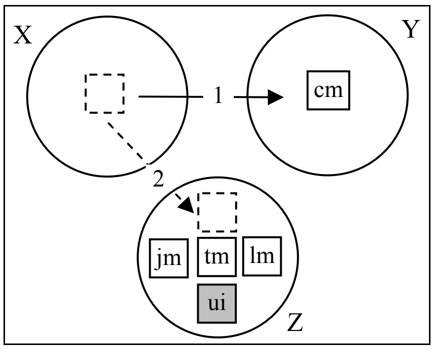


Fig. 9. Schematic diagram for model S'_1

Fig. 8 shows the complete algorithm trace for model S'_1, while Table 5 shows the efficiency metrics for the three states described above. As expected, after the environmental change and subsequent adaptation, the execution of the adapted application performs better than the non-adapted application, and in fact even better than the initial execution, while network usage becomes more balanced, resulting in improved overall efficiency.

Note that in this scenario it was not possible to change the outcome of the algorithm by varying either the weights or the parameter *r,* since the initial state provides the algorithm with only two migration options, as can be seen in Fig. 9. Since the option to move [cm] to [Y] scores better for both of the efficiency indicators I_{NU} and I_{RT}, the decision scores S'_1 to S'_5 were always higher for the decision to move [cm] to [Z] and thus tuning the algorithm based on different values did not have an effect on the outcome in this case. This is not problematic since this scenario has both demonstrated the ability of the adaptation algorithm to improve efficiency in response to a change in network utilisation, and shown that where there is a clearly preferential decision, the weighting and tuning parameters will not necessarily affect the adaptation outcome.

6. Conclusions

This paper has proposed and empirically evaluated a local adaptation strategy for distributed mobile applications, which is based on a series of formal models and a suite of metrics for mobile applications introduced by the authors in a recent paper.

The experiments demonstrated a potential practical application of the local adaptation algorithm using a number of distinct scenarios involving runtime changes in processor, memory and network utilisation. Improvements in the efficiency of the adapted distributed application were demonstrated, in comparison with the non adapted state, in response to these changes in resource utilisation. The adaptation process itself involved the rearrangement of object topology, achieved by dynamically moving objects between nodes, based on metrics that characterised: 1) The software components of the application in terms of attributes such as method invocation overhead, method intensity, size of serialised parameters, serialised object size etc. 2) Efficiency in terms of performance (response time), and resource utilisation attributes representing the percentage utilisation of specific resources such as processor, network or memory; and 3) Algorithm tuning parameters allowing preference to be given to specific attributes and their indicators, as well as the decision to favour higher attribute values over lower ones.

The authors believe that the application of the local adaptation algorithm, as well as the new set of metrics upon which it is based, are both novel and show significant potential thus serving as a significant contribution. Nevertheless, there are a number of limitations of the present study, as well as related topics that are beyond the scope of this paper, which can serve as the basis for ongoing work.

Firstly, although this paper studies a real application running on an existing framework for distributed mobile applications, future work could look at a larger example application using an increased number of nodes and a more diverse range of adaptation scenarios.

Secondly, the overhead of the metrics collection and evaluation process was not explicitly considered and thus although it was demonstrated that the adaptation algorithm could produce favourable results, future work will examine the optimisation of the collection and evaluation process so as to not offset the gains achieved through adaptation via object mobility. This will be done by integrating the metrics collection strategies and the local adaptation algorithm into the MobJeX framework, thus providing a test bed for further optimisation of this process.

Thirdly, while this paper has considered efficiency in terms of performance, and memory, network, and processor utilisation; there are other efficiency factors that could be considered such as physical storage, and financial cost where for example, contrary to a fixed local area network, a 3G or wireless network could incur a per data unit charge. Furthermore, power consumption, especially in terms of battery usage in mobile devices, is another important area for future study since the decision to favour the utilisation of the previously considered resources can have a direct impact on power consumption [27].

Finally, it should be noted that similar metrics and indicators could be specified for other quality attributes (and their sub-attributes) such as reliability [23], in which case adaptation could be performed based on more than one quality attribute.

7. Acknowledgements.

This work is part of the Applications Program of the Australian Telecommunications Cooperative Research Centre: `http://www.atcrc.com/`.

8. Appendix

Table 6. Summary of the Relationships among Metrics

Attribute		Metric	Unit	Related to
Code	Object Compilation Volume	Executable Code Size (ECS)	byte	MCT, NU
	Object Serialisation Volume	Serialised Object Size (SOS)	byte	MIT, NU
	Object Memory Volume	Object Memory Size (OMS)	byte	MU
	Method Execution Volume	Execution Memory Size (EMS)	byte	MU
	Method Body Intensity	Number of Executed Instructions (NEI)	int	ET, PU
	Method Interface Volume	Size of Serialised Parameters (SSP)	byte	IT, NU
	Method Invocation Frequency	Number of Invocations (NI)	int	NU, PU
Efficiency	Method Execution Cost	Method Execution Time (ET)	ms	-
	Method Invocation Cost	Method Invocation Time (IT)	ms	-
	Object Migration Cost	Migrate Instance Time (MIT)	ms	-
	Class Migration Cost	Migrate Class Time (MCT)	ms	-
	Network Utilisation	Network Usage (NU)	byte	IT, MCT, MIT
	Memory Utilisation	Memory Usage (MU)	byte	-
	Processor Utilisation	Processor Usage (PU)	int/s	ET

$$rt_d = \sum\nolimits_{i=1..m} \left[\mathbf{NI}_i * (rt_i^C - rt_i^D)\right] - mot, \text{ with } mot = \mathbf{MCT} + \mathbf{MIT} \text{ and } rt = \mathbf{IT} + \mathbf{ET} \quad \mathbf{(5)}$$

$$ru_d = \left|\frac{ru^C}{rc^C} - \frac{ru^D}{rc^D}\right| - \left|\frac{ru^C - ru^O}{rc^C} - \frac{ru^D + ru^O}{rc^D}\right|, \text{ with } ru^O = ru^F + \frac{\sum_{i=1..m}(ru_i * \mathbf{NI_i})}{m} \quad \mathbf{(6)}$$

where mu^F = **OMS,** mu_i = **EMS,** nu^F = **SOS** + **ECS**, nu_i = **SSP**, pu^F = 0, pu_i = f(**NEI**), rc^C and rc^D = resource capacity of the current and destination nodes of the object respectively, and m = number of methods [7].

9. References

1. Emmerich, W., *Engineering Distributed Objects*: Wiley. (2000).
2. Sun Microsystems. *Java 2 Micro Edition*. URL: http://java.sun.com/j2me/. [May 2005]
3. Symbian Ltd. *Symbian OS*. URL: http://www.symbian.com/. [May 2005]
4. Microsoft Corporation.*NET Compact Framework*. URL: http://msdn.microsoft.com/mobility/prodtechinfo/devtools/netcf/. [May 2005]
5. Ryan, C. and S. Perry, *Client/Server Configuration in a Next Generation Internet Environment: End-User, Developer, and Service Provider Perspectives*. In Proceedings: 2003 Australian Telecommunications, Networks and Applications Conference (ATNAC). Melbourne, Australia. (2003)
6. Ryan, C. and C. Westhorpe, *Application Adaptation through Transparent and Portable Object Mobility in Java*. In Proceedings: CoopIS/DOA/ODBASE (LNCS 3291). Larnaca, Cyprus: Springer-Verlag. p. 1262-1284 (2004)
7. Ryan, C. and P. Rossi, *Software, Performance and Resource Utilisation Metrics for Context-Aware Mobile Applications*. In Proceedings: International Software Metrics Symposium. Como, Italy: IEEE Computer Society. (2005)

8. Segarra, M. and F. Andre, *A Framework for Dynamic Adaptation in Wireless Environments*. In Proceedings: Technology of Object-Oriented Languages and Systems: IEEE. p. 336-347 (2000)
9. Aziz, B. and C. Jensen, *Adaptability in CORBA: The Mobile Proxy Approach*. In Proceedings: International Symposium on Distributed Objects and Applications: IEEE Computer Society. p. 295-304 (2000)
10. Fox, A., et al., *Adapting to network and client variation using active proxies: Lessons and perspectives.* IEEE Personal Communications. **5**(4): p. 10-19 (1998)
11. Noble, B., *System Support for Adaptive, Mobile Applications.* IEEE Personal Communications. **7**(1): p. 44-49 (2000)
12. Garti, D., et al., *Object Mobility for Performance Improvements of Parallel Java Applications.* Parallel and Distributed Computing. **60**(10): p. 1311-1324 (2000)
13. Ben-Shaul, I., et al., *Dynamic Self Adaptation in Distributed Systems*. In Proceedings: Self-Adaptive Software: First International Workshop. Oxford: Springer. p. 134-142 (2000)
14. Blair, G.S., et al., *A principled approach to supporting adaptation in distributed mobile environments*. In Proceedings: Software Engineering for Parallel and Distributed Systems. International Symposium on: IEEE. p. 3-12 (2000)
15. Capra, L., W. Emmerich, and C. Mascolo, *CARISMA: context-aware reflective middleware system for mobile applications.* Software Engineering, IEEE Transactions on. **29**(10): p. 929-945 (2003)
16. Chang, F. and V. Karamcheti, *Automatic configuration and run-time adaptation of distributed applications*. In Proceedings: Ninth IEEE International Symposium on High Performance Distributed Computing. Pittsburg, Pennsylvania. p. 11-20 (2000)
17. Moura, A., et al., *Dynamic support for distributed auto-adaptive applications*. In Proceedings: Workshop on Aspect Oriented Programming for Distributed Computing Systems. Vienna, Austria: IEEE. p. 451-456 (2002)
18. Silva, F., M. Endler, and F. Kon, *Developing Adaptive Distributed Applications: A Framework Overview and Experimental Results*. In Proceedings: CoopIS/DOA/ODBASE (LNCS 2888): Springer. p. 1275 - 1291 (2003)
19. Vanegas, R., et al., *QuO's runtime support for quality of service in distributed objects*. In Proceedings: International Conference on Distributed Systems Platforms and Open Distributed Processing. The Lake District, England: Sringer. p. 207-224 (1998)
20. Venkatasubramanian, N., C. Talcott, and G. Agha, *A Formal Model for Reasoning About Adaptive QoS-Enabled Middleware.* ACM Transactions on Software Engineering and Methodology. **13**(1): p. 86-147 (2004)
21. Maia, R., R. Cerqueira, and N. Rodriguez, *An Infrastructure for Development of Dynamically Adaptable Distributed Components*. In Proceedings: CoopIS, DOA, and ODBASE (LNCS 3290). Larnaca, Cyprus: Springer-Verlag. p. 1285-1302 (2004)
22. Jing, J., A. Helal, and A. Elmagarmid, *Client-Server Computing in Mobile Environments.* ACM Computing Surveys. **31**(2): p. 118-157 (1999)
23. ISO/IEC, *Information Technology - Software Product Quality - Part 1: Quality Model*. 2001, International Standards Organisation: Geneva.
24. Gilb, T., *Software Metrics*. Massachusetts: Winthrop. (1977).
25. Dujmovic, J., *A Method for Evaluation and Selection of Complex Hardware and Software Systems*. In Proceedings: International Conf. on Resource Management and Performance Evaluation of Enterprise Computer Systems. Turnersville, N.J. p. 368-378 (1996)
26. Olsina, L. and G. Rossi, *Measuring Web Applications Quality with WebQEM.* IEEE Multimedia. **9**(4): p. 20-29 (2002)
27. Chen, G., et al., *Studying energy trade offs in offloading computation/compilation in Java-enabled mobile devices.* Parallel and Distributed Systems, IEEE Transactions on. **15**(9): p. 795-809 (2004)