SPADE - Scheduler for Parallel and Distributed Execution from mobile devices

João Nuno Silva, Luís Veiga, Paulo Ferreira
INESC-ID / Technical University of Lisbon
{joao.n.silva, luis.veiga, paulo.ferreira}@inesc-id.pt

ABSTRACT

Mobile computing devices, such as mobile phones or even ultra-mobile PC's, are becoming more and more powerful. Because of this fact, users are starting to use these devices to execute tasks that until a few years ago would only be executed on a desktop PC, e.g. picture manipulation, or text editing. Furthermore, these devices, are by now almost continuously connected, either by Wi-Fi or 3G UMTS links. Nevertheless power consumption is still a major factor on these mobile devices usage, restricting autonomy.

While users should be able to employ mobile computing devices to perform these tasks with convenience, it would improve performance and reduce battery drain if the bulk processing of such tasks could be offloaded to remote hosts accessible by the same user. To accomplish this, we present SPADE, a middleware to deploy remote and parallel execution of some commodity applications to solve complex problems, from mobile devices, without any special programming effort, and by simply defining several data input sets.

In SPADE, jobs are composed of simpler tasks that will be executed on remote computers. The user states what files should be processed by each task, what application will be launched and defines the application arguments. By using SPADE any user can, for instance, accelerate a batch image manipulation by using otherwise idle remote computers, while releasing the mobile device for other tasks.

In order to make SPADE usable by a wide set of computer users we implemented two ideas: i) the execution code is a commodity piece of software already installed on the remote computers (e.g. image processing applications), and ii) the definition of the data sets to be remotely processed is done in a simple and intuitive way. The results are promising as the speedups accomplished are near optimal, while reducing power consumption, and SPADE allows the easy and efficient deployment of jobs on remote hosts.

Categories and Subject Descriptors

C.2.4 [**Computer Communication Networks**]: Distributed Systems

General Terms

Computation offload, task scheduling, mobile devices

Keywords

Mobile computing, computation offloading, bag-of-tasks

1. INTRODUCTION

Mobile devices are becoming so powerful that some tasks that previously were confined to desktop computers, are now done while roaming. Mobile smart-phones are now powerful enough to execute real operating systems (e.g. Windows Mobile, iPhone OS), allowing the execution of some productivity tools (e.g., text editing, image manipulation). Laptop computers are also becoming more compact while having its computing power increased. MID [6] devices or other ultra-mobile devices, having display from 7 inches are now capable of executing full featured desktop operating systems (Linux or Windows).

Another characteristic of these devices, also borrowed from desktop computers, is network connectivity. Due to 3G UMTS and Wi-Fi wide availability, these devices can now be almost constantly connected to the Internet. Such connections are now cost effective and have speed on the order of Mbit.

Even though these devices can execute most desktop applications, their execution speed is lower than on their desktop counterparts and drain more power from batteries than less demanding applications. While batteries are becoming more powerful, these new usage patterns reduce the effective autonomy these devices have.

It would be good to still allow users to run such desktop applications on their remote devices, while reducing power usage and also reducing application execution time. This can be accomplished taking advantage of, otherwise idle, remote computers, also owned by the mobile user. The communication between the mobile device and the remote desktop computer may use either Wi-Fi or 3G network links.

With SPADE, we present a tool that allows the remote execution of common computing tasks on several remote computers, otherwise executed on mobile devices. Users should be able to speed the execution of tasks such as image manipulation by taking advantage of remote idle computers, releasing the mobile device to other tasks and reducing power consumption. We propose the application of techniques similar to the ones used in Grid and Cluster computing, while taking into account the particularities of mobile users.

On the area of scientific computing the offloading of computations to more powerful remote computers is a good solution to speed lengthy jobs. Users are required to use specific pre-installed applications. On a desktop environment, a user owning several computers can install a cluster. The XGrid middleware allows easy orchestration and job distribution between computers. On mobile devices, if a user wished to run applications with desktop speed, he would have to initiate the computation on the remote computer using, for instance, VNC. In this case, the data should be, not in the mobile device, but already on the remote desktop computer. The offloading of computations to remote hosts has been proposed ealier, but involved the evaluation and adaptation of application source code.

Remote execution of generic software packages may seem impractical and without standard use, but there are several applications whose data can be easily partitioned and several instances of the application executed in parallel. The scope of commodity software that can be executed in a more powerful remote environment (single computer or cluster) ranges from ray tracing systems (such as POV-Ray) to video or image processing (ImageMagick) or even statistical packages. For instance batch image manipulations using the ImageMagick package can be executed on another computer, as long as it has the same software installed. The user defines the arguments to apply to each image conversion, SPADE uploads the images to a remote computer and invokes the installed ImageMagick application there, conveying the arguments defined by the user. The resulting files are downloaded after the conversion is complete.

In this paper we present SPADE, a system that facilitates the execution of commodity software on remote idle computers in order to reduce the time it would take on a mobile device, while reducing power consumption. These jobs should be executed by commodity applications, invocable from a command line. They can be composed by a single lengthy task or by multiple tasks whose parameters differ on only a numerical values or file. The owner of the CPU providers (and mobile device) must register their applications (e.g., R, POV-Ray) on the SPADE system. When a mobile device has a complex job to be executed on remote computers, the user supplies SPADE with the different input files and the parameters that should be provided to that task. SPADE is responsible for uploading the necessary input files to the remote host, executing the selected application and downloading the result file. If the user has a batch of files to be processed, SPADE can also be used. The user only needs to supply a generic command line that is parameterized to each task. Such command lines can differ on the input file or on a numerical value. SPADE will upload the files, with the corresponding command line, to the remote available computers. After processing, resulting files are downloaded back to the mobile device, possibly on-demand.

The definition of usable applications and jobs, and how they are possibly divided, is made using a simple user interface, allowing any user with minimum computer knowledge to do it. The selection of the available hosts, where remote tasks are executed, is performed in a easy transparent way.

In the next section, we present current systems that try to provide a way to use remote spare cycles. In section 3, we enumerate the requirements a cycle sharing system must comply with in order to become useful and widely used. In Section 4, we describe the SPADE architecture, its implementation and execution. In the last sections we present SPADE evaluation and draw some conclusions.

2. RELATED WORK

In the area of cluster computing, some developments have been made on middleware to allow easy deployment of applications over several computers. One of those tools is Xgrid [1]. Xgrid allows the use of several computers as a cluster, providing a tool to launch and distribute parallel computations. Even though XGrid allows an easy distribution of work, for each job, users have to write a configuration file with corresponding parameters. There is no way to define generic parameters that are later instantiated to each os the tasks.

On a grid, schedulers like Condor [8] or GLOBUS [5] allow launching of lengthy jobs on remote computers, users write a launching definition file and the middleware is responsible for selecting the best available hosts to execute the tasks. In order to use remote infrastructures users are required to have the necessary authorizations and credentials. Ganga [4] is a python application that allows graphical definition of the tasks to be executed on a remote cluster. Users either define a single lengthy task (processing application and parameters), or define all tasks that compose the jobs. When defining several tasks, users must supply different parameters to each one. There is no way to define a generic set of arguments and parameterize them to each task.

Grid infrastructures have strong authentication mechanisms that are required on a public conventional installation, having users with different requirements and roles. On a simple environment where all computers (mobile device, from where tasks are launched, and remote computers) are owned and administered by the same user, such a heavy solution is over demanding. Implementing a simpler authentication method instead, security would still be guaranteed with limited overhead.

Mobile grids may be a solution to the problem of offloading lengthy tasks from mobile devices . Recent work on mobile grids, such as Mobile OGSI.NET [3] or Akogrimo [11], has been centered on the integration of services with mobile devices. Their aim is to develop infrastructures to help the deployment of distributed applications running on mobile devices and accessing remote services.

The approach used to develop mobile computing systems, consisted on splitting the application on components (e.g. ICrafter [9]). The user interface runs on the mobile device and the computing intensive parts running on a remote computer. This approach reduces the burden on the mobile device but requires the application to be specially developed targeting the mobile environment.

Process offloading has already proved to reduce power consumption [10], but required the analysis and adaptation of application source code as proposed by Zhiyuan Li [7]. The system proposed performs code analysis in order to discover functions that would be executed faster if offloaded to a remote host. Rajesh Krishna Balan [2] presents a language to simplify the adaptation of applications, so that they can be easily offloaded from mobile devices, to allow cyber foraging. The presented solution is composed of: i) a language to define how the application should be split; ii) a middleware for management of every distributed application; and iii) stub generators for the modules that should be offloaded. With these modules, programmers can modify any application so that part of it can be executed on a remote host.

In summary, even though such code offloading solutions provides good results, they all require modifications at the application source code level.

Presently, no tool allows the efficient execution of unmodified commodity applications from mobile devices, taking advantage of remote devices, to speed them and reduce power consumption.

3. SPADE REQUIREMENTS

As a tool to offload computations to remote computers, SPADE is most useful to those who have lengthy tasks to perform: lengthy single task jobs, repetitive data analysis, or long processing of large batches of files. Users with such requirements range from scientists who need to process batches of data, to the hobbyist who needs to create thumbnails for his collection of photos, or to designers who need to generate a detailed image using ray tracing tools. All these users have in common the fact that they are literate in some sort of tool (statistical package, image manipulation tool or image generator) but are not proficient in programming parallel or distributed applications.

The jobs we propose to accelerate, by executing them on remote machines, should have data that is naturally partitioned: the scientist and the hobbyist have their data or images in different files, while the artist knows how to tell the ray tracer to generate only a small sub-set of the picture.

Without the existence of remote computers to host their processing, the data files that are present on the mobile device would be processed slowly or would have to wait until the user had direct access to a desktop computer. Besides usability, in order for SPADE to be useful it is necessary that whenever a user has a job to complete, there are enough remote hosts willing to provide their idle CPU cycles. In order to accomplish this, it is only necessary that applications to be used are installed on desktop computers owned by the mobile user.

3.1 Applications

One of the most relevant factors for the success of a system like SPADE is the amount of applications that can be executed in it. Such applications exist in most areas: statistical analysis, simulation, image processing, image or video generation.

R is a language environment for statistical computing and graphics. This environment processes scripts written in the R language. SPADE is useful in situations where a user has several scripts to execute in the R environment or has a script that must evaluate several data sets. In the R command line interface, a user can define the input and output files as well as other R variables, allowing the easy interaction of the SPADE system with the R program.

ImageMagick packages a set of image manipulation tools that run with the same interface (command line arguments) on most desktop operating systems. When a user wants to apply the same transformation (for instance, color correction, or border generation) to a batch of images, one of the ImageMagick applications would have to be executed for each file. A non-expert user would sequentially process every image while SPADE would execute each image transformation on different hosts in parallel.

POV-Ray is a ray tracing image generator that processes text files describing 3D scenes and generates the corresponding image. POV-Ray may also generate a sequence of images that can be used to create a movie. It runs on a variety of operating systems delivering the same results on every one and has shell commands for the definition of the viewport or the timestep to be rendered. Image rendering may take advantage of a system such as SPADE, by distributing the rendering of small parts of the final image on different computers, while a movie generation could also be accelerated by rendering each frame in parallel.

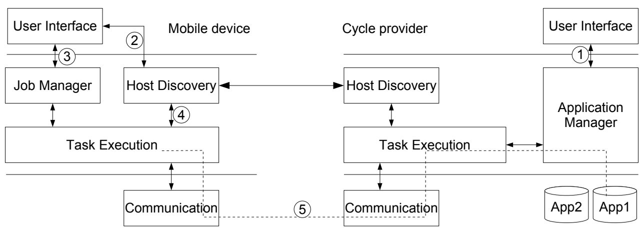


Figure 1: SPADE System Architecture

The LaTeX typesetting language can also be used through SPADE. The user supplies the input files (source, and image files) that are processed on a remote host. Thus speeding the compilation.

Obviously, other tools such as audio or video processing tools can also be safely used with SPADE.

3.2 Input Data Definition

Besides the definition of jobs with only one task, SPADE user interface allows the definition of jobs with multiple tasks, each one possibly processing different data. We present some examples of data partition requirements most SPADE target users have:

- Each task processes a different input file, also generating a different output file. This category includes batch image processing or statistical analysis of different data sets.
- Each task has a numerical argument ranging from 0 to N. This numerical value can be a task identifier or its transformation. In the case of a movie generation with POV-Ray, this numerical argument may represent the timestep of the frame being generated within the movie.
- Each task has several numerical arguments generated as a function of the task identifier. The rendering of a complex image can be parallelized giving to each task the responsibility to generate a piece of the output image. For instance, a task would render a view from $(0, 0)$ to $(1023, 511)$ while another task would generate the view from $(0, 512)$ to $(1023, 1023)$. This example represents a two-dimensional space sweep but a one-dimensional space sweep should also be possible.

Any combination of these examples could be used to define a job task.

4. SPADE SYSTEM

A SPADE daemon must be running in each cycle provider (desktop computer), on the mobile device (consumer computer) only the client needs to be executing. Their architecture is presented in Fig. 1.

As stated earlier, SPADE relies on commodity software installed on the cycle providing computers. It is the responsibility of the *Application Manager* to keep a record of the software accessible by remote SPADE clients. Before receiving a processing task, the computer owner must register the application in order to make it available do SPADE (step 1 on the Cycle Provider). The user has to provide the path to the application and its well know name (`latex`, for instance). Whenever there is a task to be executed, this module is contacted to provide the path to the executable.

Before submitting any job from the mobile device, the user must register one desktop that will later be responsible for executing those jobs (step 2). Other available remote computers can be added manually or discovered through an already registered computer.

The user provides the *Job Manager* module with all information regarding the job to be executed over the network (step 3). This information includes the application's well-known name, its arguments and the files (input or output files) that should be transmitted over the network. The user must also state how many tasks form the complete job. Then, this module submits each task to the *Task Execution* module and keeps track of every task state (*not started*, *started* or *finished*). The *Task Execution* module chooses where to execute each task (step 5). Remote computers are selected from the information provided by the *Host Discovery* module (step 4). The *communication* module is responsible for all communication and data transfer between SPADE daemons.

4.1 SPADE Implementation

SPADE was implemented in Python; the XML-RPC library was used to perform all communication between SPADE daemons (but any other programming language that provided some sort of RPC mechanism could have been used). In Fig. 2, we present the UML class diagram of an executing daemon.

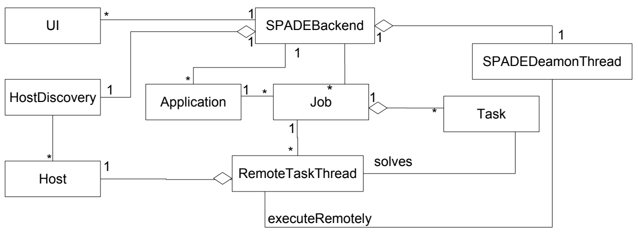


Figure 2: UML class diagram

SPADEBackend is the class that encapsulates all SPADE daemon behavior. It receives requests from the user interface module to register new applications and new jobs. *SPADEBackend* has a list of *Applications* containing the well known name of each application and the path to the corresponding executable.

Each *Job* object stores the corresponding application and the set of tasks that make it. Each *Task* object contains all the information necessary to its local or remote execution: arguments, input files data and output file name.

The *UI* class handles all user interaction. This interface was developed using *wxwidgets*, making it compatible with several operating systems.

On every cycle provider, there is a *Host Discovery* module that is responsible for the discovery of other daemons on the same network. It was implemented with a simple class (*HostDiscovery*) that performs network broadcasts in order to find other SPADE daemons. On the mobile device the *Host Discovery* module contacts the remote hosts whose identifications were provided by the user, in order to fetch the addresses of all the other cycle providers discovered .

In order to allow parallel execution of several tasks, for every remote cycle provider that has the required application, a remote *RemoteTaskThread* thread is created. Each *RemoteTaskThread* contacts the corresponding SPADE daemon, delivering the input files and the shell command and receiving the result file.

4.2 Job Submission

Before any job submission to SPADE, it is necessary that, on the cycle providers, the user has previously registered the applications that can be executed. This is accomplished by providing the location of the executable and a name.

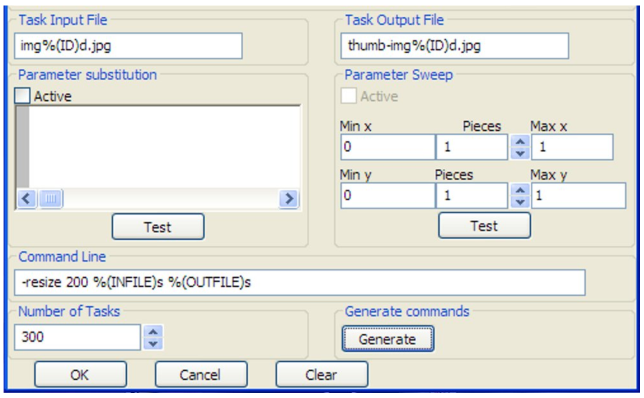
In order to define the jobs that will be executed, the user must first select the necessary application, the location of the input files and what files should be transmitted to the remote hosts. The user then has to fill the form presented in Fig. 3 to define the command line of each task.

When filling the *Command Line*, *Task Input File* and *Task Output File* the user can use `%(ID)d` placeholder to represent the task identifier. Later, for each task this placeholder will be replaced with the correspondent task identifier. In the *Command Line* field the user can also use the `%(INFILE)s` or `%(OUTFILE)s` placeholders that will later be replaced by the actual input and output file names. These placeholders allow the definition of a different command line for each task but are not flexible enough.

In example presented in Fig. 3 the actual arguments for each task would be:

```
-resize 200 img0.jpg thumb-img0.jpg
-resize 200 img1.jpg thumb-img1.jpg
...
-resize 200 img299.jpg thumb-img299.jpg
```

In order to allow the easy definition of a two-dimensional space sweep, the user can define the limits of that space



Task Input File	img%(ID)d.jpg
Task Output File	thumb-img%(ID)d.jpg
Parameter substitution: Active	☐
Parameter Sweep: Active	☐
Min x	0
Pieces	1
Max x	1
Min y	0
Pieces	1
Max y	1
Command Line	-resize 200 %(INFILE)s %(OUTFILE)s
Number of Tasks	300

Figure 3: Job submission user interface close-up

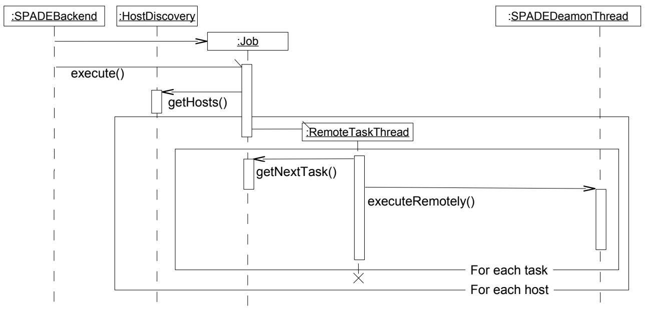


Figure 4: Job execution UML sequence diagram

(*Min X*, *Max X*, *Min Y* and *Max Y* and on how many slices that space will be split in. On the *Command Line*, *Task Input File* and *Task Output File* fields the user will use the `%(MINX)s`, `%(MAXX)s` `%(MINY)s` or `%(MAXY)s` placeholders to represent the limits of the space that each task is responsible for. In the *Parameter Substitution* field the user can write a simple function with a `transFunc(i, nTasks)` interface, where `i` will be the actual task identifier and the `nTasks` is the total number of tasks. For each task identifier, the returned value will replace the `%(NEWPARAM)s` placeholder. Then, actual values of each placeholder will be calculated for each task, guaranteeing that all the parameter space is covered.

4.3 Job Execution

Whenever a job is created by means of the form presented in Fig 3, an object of class *Job* is created and populated with all its tasks; each object of class *Task* will have the necessary information: command line and input and output files. In Fig. 4, we present a Job execution UML sequence diagram. After the *Job* object creation and initialization, a thread is created. This thread will get a list of hosts (`get_hosts()`) that have SPADE installed and the corresponding job application . Then, for each host discovered, a *RemoteHostThread* is created. Each one of these threads will fetch tasks (`getNextTask()`) and send the necessary data to the remote host (`executeRemotely()`).

In order to execute the task, the remote host must receive the following information: the name of the input and output files, the task identification and the command line without the placeholders replaced. Besides this, the actual contents of the input file should also be transferred to the remote host.

After receiving all information and data the *SPADEDeamonThread* creates a temporary directory and copies the input file there. The actual command line is built taking into account the executable location and the temporary directory where the input file is. The placeholders `%(INFILE)s` and `%(OUTFILE)s` are replaced with the concatenation of the temporary directory and the files names. After the generation of the actual command line the task is executed, the result output file is read and its contents returned to the cycle consumer host.

Every task may be in one of three possible states: *not started*, *started* and *finished*. When a task is executed its state changes from *not started* to *started*, until receiving the result of its execution, the state of a task remains in the *started* state, changing to *finished* after the reception of its results. In order to guarantee that all tasks are executed, after all tasks are initiated (no more tasks in the *not started* state) the previously started tasks are sent to available remote hosts. This way, duplicate results may be received, but there is the guarantee that all tasks will finish.

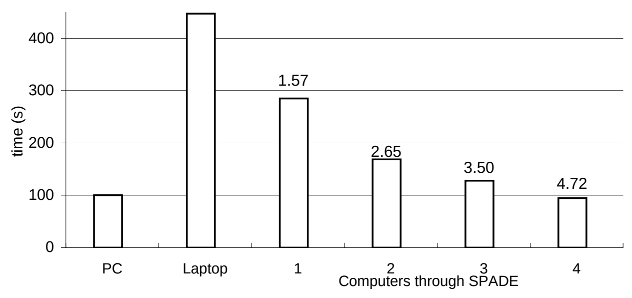


Figure 5: Job execution time (values on top of bars represent speedups relative to Laptop time)

4.4 SPADE Evaluation

In order to evaluate the usability of SPADE and the possible speedups we used SPADE to create thumbnails for 300 1600x1200 pixel JPG images, each one with 260Kb of size. Each thumbnail is generated by executing the *convert* utility from the ImageMagick suit, from this resulted a 4Kb file. The mobile system used consisted on a eeePC laptop with a 900 MHz Intel Celeron Mobile processor and 512 Mbyte of memory running the Xandros Linux distribution. As cycle providers (remote computers) we used Pentium 4 at 3.2 GHz with 1 Gbyte of RAM computers running Microsoft Windows XP. The laptop was connected to the cycle providers by a Wi-Fi wireless link and 100Mbit network.

The time to sequentially convert all 300 images on the mobile device is 450 s, on a desktop computer these conversions took 100 seconds. The same job executed within SPADE but on a single remote computer took about 280 s. For this job the data transmission overhead is about 180 seconds. In this extreme example the transmission time is almost twice the processing time on a remote computer. Despite this, even from the use of one remote computer substantial gains are obtained. By using more remote hosts, the gains are more noticeable as shown in Fig. 5 (speedup of 2.56 with only two computers).

The offloading of jobs to remote computers also brings gains on power consumption and battery duration. In order to evaluate this, after fully charging the batteries, we executed the maximum of thumbnails generations, until battery depletion. This way we can evaluate what method (local or remote processing) spends less energy.

	Tasks	Duration (m)
Idle		184
Local	7560	160
SPADE	9467	175

Table 1: Battery capacity

On table 1 we can observe that while executing the tasks remotely, more tasks were executed, meaning that each remote tasks spent less energy than the local counterpart.

The evaluated example is extreme (high data transmission time versus remote processing time ratio), but good results were obtained. On other scenarios, the results (speedups and consumed power) can be even better.

With respect to usability factors, the job creation interface is sufficient for most jobs a common user may have. The user is capable of defining jobs, where each task receives as parameter the task identifier and where different files are processed by each task as well as easily define a one or two-dimensional space partition. The writing of the Parameter substitution function requires some programming knowledge, but by providing some transformation function, any user can adapt them to its needs.

5. CONCLUSIONS

We presented a system that offloads long computations from mobile devices to remote computers, without requiring any special programming skills from the user. This way, it is possible to free the mobile device to other tasks, speed the execution of those tasks and reduce the required power consumption.

These tasks are those solved by commodity software that can be invoked from the command line. These software packages should be installed on several remote computers, owned by the mobile computer user. SPADE speeds a job's execution by sending the data to one or several remote computers. Each data file is processed on a remote computer. The definition of these data files and every task parameter is done in a simple but efficient way. The user neither needs to know how to program distributed or parallel applications, nor has to deploy and install a centralized complex infrastructure.

Our main contribution is the use of a mechanism that allows easy offloading of time and power consuming tasks from mobile devices to remote idle computers. From our results even short tasks can be executed on remote hosts with gains: i) the remote tasks total execution times (data transmission and computation) are lower than if computed on the local device, and ii) the electric power necessary to transmit the data to a remote host is lower than the used in a local computation. Taking this into account, SPADE can be a viable solution to the execution of jobs from mobile devices.

If connected by a slower network link, speedups can not be as good. Transmission of batches of tasks and results can overcome the network link speed. Instead of sending one task at a time to each remote computers, the distribution of tasks can be performed in two steps: i) sending tasks in a batch to one remote computer, ii) distributing tasks to worker computers through a local area network link. The reception of results can also be performed in a similar way: after a task finishes, its result is sent to the computer responsible for aggregating them; later, the mobile device downloads all the results from one single remote computer.

By using remote computers owned by the mobile user, the threats associated with mobile code (both to the owner of the tasks and to the owner of the host computer) are not present: the computing environment is well known (guaranteeing the correctness of the results) as well as the code to be executed remotely (guaranteeing the nonexistence of malicious code). Even though, simple identification and authorization mechanisms should be implemented.

SPADE can be used as a metascheduler, allowing the execution of tasks on existing grid infrastructures. Jobs could be uploaded to a remote computer (owned by the mobile user) and forwarded to available computers within a grid infrastructure. The mobile device would be released from the burden of client grid software, but the user could still benefit of higher performance.

6. REFERENCES

[1] Apple Computer, Inc. Xgrid - the simple solution for distributed computing. http://apple.com/macosx/features/xgrid/.

[2] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb. Simplifying cyber foraging for mobile devices. In *MobiSys '07: Proc. of the 5th international conference on Mobile systems, applications and services*, pages 272–285, New York, NY, USA, 2007. ACM.

[3] D. Chu and M. Humphrey. Mobile ogsi.net: grid computing on mobile devices. *Grid Computing, 2004. Proc. Fifth IEEE/ACM International Workshop on*, pages 182–191, November 2004.

[4] U. Egede, K.Harrison, R. Jones, A. Maier, J. Moscicki, G. Patrick, A. Soroko, and C. Tan. Ganga user interface for job definition and management. In *Proc. Fourth International Workshop on Frontier Science: New Frontiers in Subnuclear Physics*, Italy, September 2005. Laboratori Nazionali di Frascati.

[5] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, pages 2–13. Springer-Verlag, 2005.

[6] Intel Corporation. Mobile internet devices (mids). http://www.intel.com/products/mid.

[7] Z. Li, C. Wang, and R. Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *CASES '01: Proc. of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems*, 2001. ACM.

[8] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proc. of the 8th Intl.Conf. of Distributed Computing Systems.* IEEE Computer Society, June 1988.

[9] S. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd. Icrafter: A service framework for ubiquitous computing environments. In *Proc. of the 3rd Intl. Conf. on Ubiquitous Computing.* Springer-Verlag, 2001.

[10] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning. Saving portable computer battery power through remote process execution. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2(1):19–26, 1998.

[11] M. Waldburger, C. Morariu, P. Racz, J. Jähnert, S. Wesner, and B. Stiller. Grids in a mobile world: Akogrimo's network and business views. *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, 30(1):32–43, Jan 2007.