# Offload Destination Selection to Enable Distributed Computation on Battlefields

Tamim Sookoor, David Doria,
David Bruno, Dale Shires
Computational Sciences Division
U.S. Army Research Laboratory
Aberdeen Proving Ground, Maryland 21005 USA
Email: tamim.i.sookoor.civ@mail.mil

Brian Swenson
School of Electrical and
Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250 USA
Email: bpswenson@gatech.edu

Lori Pollock
Department of Computer and
Information Sciences
University of Delaware
Newark, Delaware 19716 USA
Email: pollock@cis.udel.edu

***Abstract*—The tactical edge (battlefields, disaster relief operations, etc.) is an ideal domain for utilizing mobile devices for computation. However, these situations require jobs to complete in a timely manner. Efficiently offloading computation can ensure timely job completion, but access to cloud computing infrastructure is typically limited, unreliable, or unavailable. As an alternative, we propose using vehicle-mounted high performance computers, called Tactical HPCs (T-HPCs), as offload targets. A common scenario includes multiple offload targets with different computation capabilities, multiple clients wanting to offload computations, and wireless ad-hoc connectivity. While offloading strategies exist, currently none address the requirements of task scheduling in this setting. This paper presents an algorithm for offloading tasks from multiple sources while taking into account the computation capabilities and current utilization of available offload targets. In addition, distributed scheduling avoids a single point of failure which could be disastrous on the tactical edge. We report the results of simulations showing that our method reduces job completion time compared to alternative offloading schemes.**

## I. Introduction

The observation of declining PC shipments coinciding with an increase in smartphone sales in recent years [1] has prompted researchers to look towards the post-PC era. In an era when mobile devices such as smartphones and tablets will be the primary interfaces of the end user, shortcomings of such devices must be addressed. The two most glaring weaknesses of such devices are their limited battery life and constrained computing capabilities. One approach that is being widely advocated to address both these drawbacks is computation offloading [2], [3], [4], [5].

Unfortunately, most offloading algorithms target scenarios where they can offload computation to cloud computing services such as Amazon EC2 [6]. These services provide the illusion of infinite computation capability due to a large number of high-performance computers (HPCs) and dedicated schedulers that efficiently allocate incoming tasks to a known set of processors on a static, wired network (Figure 1). HPCs, or supercomputers, execute tasks requiring large amounts of computing power for short periods of time. Therefore, in environments with access to cloud computing, the main concern of offloading algorithms is whether the job completion time locally is less than the round-trip time to the cloud. Some solutions consider the computation time on the cloud, but require a profile based on previous executions to predict the time of execution for current tasks [3].

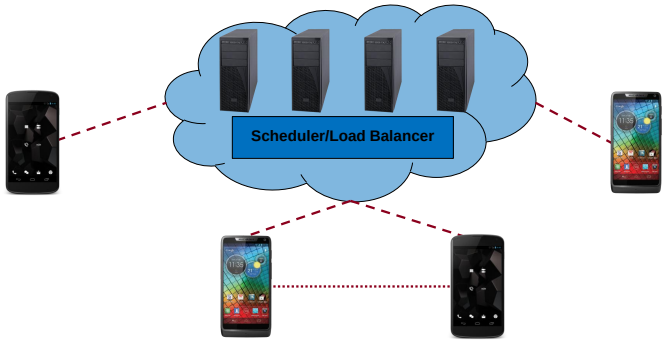


**Fig. 1.** Existing offloading scenarios where computation can be offloaded to cloud computing services or neighboring mobile devices

In domains such as battlefields and disaster areas, called the *tactical edge*, users often rely on mobile devices for computation. However, limited communication infrastructure and power resources make access to cloud infrastructure very difficult. This is a significant problem, as fast computation is essential and may save lives in these domains. To remedy this, we propose using Tactical HPCs (T-HPCs) [7] composed of vehicle-mounted HPCs as offload targets. We depict an overview of the situation in Figure 2. With the availability of multiple T-HPCs with different computational capabilities and communication channels, each offload source can split its computational job into tasks and distribute them to the available offload targets. Such an infrastructure cannot have centralized schedulers and load balancers due to the high mobility of assets and the desire to not have single points of failure. Additionally, since T-HPCs may go in and out of range, the availability of the set of offload targets can change.

In this paper, we propose a computation offloading algorithm where the clients perform job scheduling in order to make their offloading decisions. We describe how the algorithm can leverage awareness of the computation capabilities and current utilization of offload targets to improve offloading decisions. We evaluate the benefit of utilization-awareness on total job completion time by comparing our approach with alternative distributed offloading algorithms.

We base our approach on the hypothesis that in Mobile

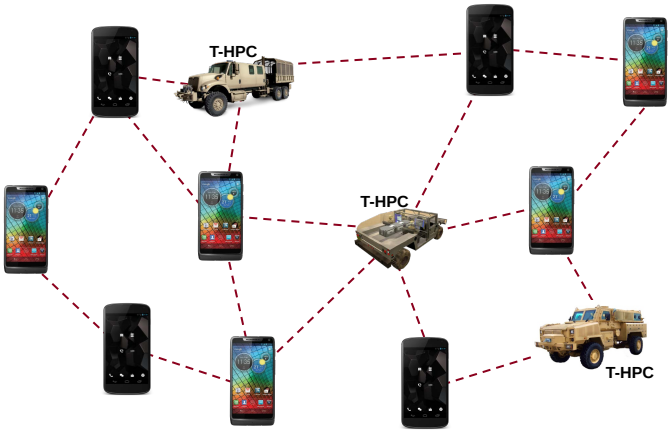


**Fig. 2.** Mobile Ad-Hoc Network of Heterogeneous networks composed of smartphones and T-HPCs

Ad-Hoc Networks (MANETs) composed of mobile devices and heterogeneous T-HPCs, considering the computation load on the HPC assets when making offloading decisions would result in shorter job completion times. We base this hypothesis on the concern that if the current load of the offload target is not considered, mobile devices would have to offload to targets either randomly or based on other heuristics, such as hop-count, which may result in them offloading computation to overloaded devices while idle devices are present. The main contributions of this paper are:

- A distributed scheduling algorithm for computation offloading that schedules multiple tasks from each client and considers computation capabilities and current utilization of offload targets
- Simulation evaluation comparing job completion times against alternative distributed offloading strategies. Parameters varied include the number of offload sources, the number of offload targets, communication bandwidth, and job size.

## II. State of the Art

Computation offloading is a common technique for reducing computation time and conserving battery power when computing on mobile devices. Two fields of research relate to our work: mobile device offloading and distributed task scheduling. Some mobile device offloading solutions address scenarios when access to stable cloud infrastructure is lacking, but to our knowledge the effect of scheduling offloaded tasks from mobile devices with knowledge of the offload targets' current utilization has not been explored. Algorithms to efficiently schedule incoming tasks in multi-processor systems are available, however, they all rely upon a static network of computers, usually connected with reliable wired links, and a master computer that is aware of the states of all the devices and makes all scheduling and load balancing decisions. In Sections II-A and II-B, we describe examples from both areas of research and describe their applicability to our problem domain.

### A. Computation Offloading

Computation offloading is an instance of cyber foraging [8] which is the leveraging of resource-rich infrastructure for computing on mobile devices. It is currently a heavily researched approach to augmenting the limited capabilities of smartphones when used as a computing device. MAUI [4] uses the Common Language Runtime (CLR) to create two versions of applications: one that can execute on a mobile device and the other that can run on the cloud. At runtime, it decides which version to execute based on CPU utilization and communication cost. It profiles the communication cost by periodically measuring the network bandwidth and uses previous invocations of an application to profile computation gain.

CloneCloud [5] can minimize energy consumption or execution time by automatically identifying computation intensive methods and offloading them. ThinkAir [3] migrates Virtual Machines (VMs) to the cloud and lets them handle method-level offloading, using a heuristic for energy savings and runtime savings. None of these approaches decide between multiple candidate offload targets in a distributed manner or takes into account the current utilization of the multiple offload targets. These approaches also assume a stable environment with reliable access to cloud infrastructure.

Shi et al. [2] identify the challenges associated with unreliable access to cloud infrastructure and propose Serendipity [9] to address the scenario where there is no access to the cloud. It uses an opportunistic offloading scheme to leverage computation from neighboring mobile devices. Whenever it encounters another mobile device, Serendipity obtains its hardware profile and the list of tasks the device is currently executing which it uses to decide if computation should be offloaded to that device. Thus, Serendipity considers the computation load of encountered devices but it does so on a device-by-device basis as it encounters them. Serendipity targets a different application context than the algorithm presented in this paper, therefore it does not perform distributed scheduling over all possible offload targets.

A number of computation offloading algorithms consider the utilization of the offloading target, but do so in order to make the decision of whether to offload or not. MECCA [10] compares the estimated cost of running an application locally as opposed to in the cloud and offloads if the estimated gain is greater than a threshold. Flores and Srirama [11] present a fuzzy logic decision model that decides between remote processing and local processing based on a set of rules. MARS [11] uses an efficient offloading decision algorithm that is sufficiently lightweight to execute on mobile devices. This algorithm maintains a priority queue of remote procedure calls (RPCs) ordered by the estimated speedup of offloading. Then it offloads RPCs at the head of the queue if the estimated energy consumption of offloading is greater than or equal to the inverse of a Greediness parameter, $G$. It executes RPCs at the bottom of the queue in their estimated enegy consumption for local execution is less than $G$. While all of these approaches take the utilization of the offloading target into consideration, they do so in order to decide whether to offload to a single offload target. Our approach is different in that there are multiple heterogeneous offload targets, and the algorithm, in

addition to deciding whether to offload, has to also decide where to offload.

Verbelen et al. present Cloudlets [12] as a means of bringing the cloud to the mobile user by allowing the cooperation of all the devices in a LAN network. Each real or virtualized hardware device in the cloudlet is managed by a Node Agent (NA) and the cloudlet is managed by a Cloudlet Agent (CA) which has a global overview of all available resources, and thus can calculate the global optimum deployment of jobs to devices. While a CA can maintain the global state of a cloudlet within a home or an office building for instance, it would be much less efficient to do so in a larger, harsher environment such as a battlefield. Also, such an architecture introduces a single point of failure which is not desirable for military applications.

Balan et al. present [13] *tactics* as a declarative method for specifying the partitioning of applications between local and remote execution. They describe a remote execution system named *Chroma* that uses the remote execution tactics to make partitioning decisions. A declarative approach such as remote execution tactics could be used with our algorithm to handle job partitioning since we ignore this detail and assume jobs can be infinitely partitioned.

### B. Distributed Task Scheduling

Efficiently scheduling tasks in a distributed computing environment is similar to our work. Systems such as Condor-G [14] and Legion [15] provide algorithms to schedule jobs submitted by independent users in Grid platforms. Dryad [16] attempts to optimize the throughput of a distributed computing cluster by mapping a dataflow graph to the available computers. All of these approaches target fixed networks of wired devices and therefore do not satisfy the constraints associated with a heterogeneous MANET.

There are a number of areas of research related to the resource allocation problem for smartphone offloading including task scheduling for heterogeneous computer systems, task allocation for mobile robots, and resource allocation in wireless sensor networks. Briceno et al. [17], [18] describe techniques to schedule tasks arriving in a heterogeneous computing system while ensuring the robustness of the schedule. The algorithms are robust to unknown task arrival times and can satisfy various constraints such as ensuring high priority tasks are executed before other tasks, yet requires a central task allocator that is aware of all the tasks and the available resources at any point in time. Task allocation in multi-robot systems [19], [20] addresses the complexity associated with mobility yet do not have the constraint of a limited battery life that is faced by a resource allocator on a smartphone. Mainland et al. [21] present an approach to allocate resources in a wireless sensor network in a decentralized manner. In the event of no connectivity to HPC assets, we propose using a scheme similar to that presented by Arslan et al. [22] or Shi et al. [9] to offload computation to other smartphones. This section describes some of this work and how they relate to the task allocation problem we are addressing. We also point out why the existing solutions are insufficient for our needs.

Many methods have been proposed to allocate resources to processes, such as assigning processors to tasks or people to jobs, where a single entity, such as a server, is responsible for scheduling the resources. Young et al. [23] describe an algorithm to schedule tasks on a heterogeneous cluster of machines in order to meet as many task deadlines as possible without violating a system energy constraint. Briceno et al. [17] present a static load balancing scheme for the satellite data processing portion of a space-based weather monitoring system. Kuhn [24] presents a solution for the assignment problem using the work of two Hungarian mathematicians: D. Konig and E. Egervary. While some of the ideas presented in these papers may be applicable in scheduling applications once they are queued on an HPC asset, they do not consider many of the issues that arise in a tactical cluster of smartphones and cloudlets.

Lim et al. [25] tackle the problem of allocating the limited sensing, processing, and communication resources in a wireless sensor network, without a central coordinator, in order to minimize costs and maximize network capability. Mainland et al. [21] present Self-Organizing Resource Allocation (SORA), a novel algorithm to allocated limited resources on sensor nodes in order to maximize the nodes contribution to the network. ADRA highlights the complications in distributed resource allocation and presents an approach where simple local actions performed by nodes gives rise to a desired global behavior. SORA presents an interesting approach to distributed resource allocation. While it is used to help nodes decide which actions to take at any point in time, a similar virtual market approach can be used to allow smartphones to select HPC assets to which tasks should be offloaded.

Arslan et al. [22] present a distributed computing infrastructure, called Computing While Charging (CWC) that uses smartphones to run tasks that traditionally executed on servers. Shi et al. [9] describe Serendipity which is a framework to enable the offloading of applications from smartphones to other mobile devices. CirrusCloud [2] extends the ideas of Serendipity, which targets offloading only to other smartphones, to a generalized cyber-foraging platform. CWC requires a centralized server to schedule tasks on phones which will not be feasibile in a military scenario. Also, offloading in a tactical environment should occur when phones are being used not when they are charging. Serendipity and CirrusCloud provides a scheduler that could be extended for offloading to HPC assets.

## III. Utilization-aware Offloading

Our goal is to minimize job completion time by offloading computation from mobile devices. In an environment where there is no centralized scheduler to efficiently allocate tasks to the offload targets, the offload sources have to intelligently schedule the offloaded tasks to the available targets. In this section, we describe how mobile devices obtain the utilizations of the offload targets and the state of the network connections. We use this information to decompose a job into tasks and assign the tasks to the collection of offload targets.

### A. Scheduling Offloaded Computation

To determine how much of the job to offload to each potential offload target, we pose the job division as an integer programming problem. The goal is to divide a job $J$ (expressed as a number of operations) into $n$ tasks.

To describe our problem formulation, we must start by introducing some notation. We denote the size of the task (in operations) given to offload target $i$ as $j_i$. Then, we compute the run time of each task as $r_i = \frac{j_i}{c_i}$ (in seconds). The task completion time, $t_i$, is the time that it takes from the point that the offload source wants to schedule the task to the time that the offload target returns the result of the task. This is a key part of the formulation. The parameters we have chosen to include in the model are:

- The capability $c_i$ (with the units as operations per second $\frac{ops}{sec}$) of the assigned offload target.
- The size of the task $j_i$ (with units of operations).
- The channel bandwith $W$ (with units of bytes per second $\frac{B}{sec}$).
- The current task queue on the offload target $Q$ (with units of seconds). The offload target keeps a queue of tasks, only executing one at a time (modeling a single thread) in a first come, first served order.
- The constant $R$ defines the ratio of the number of bytes that an offload source wants to transmit to the number of operations required (with units of bytes per operation $\frac{B}{ops}$). This ratio $R$ is application specific. For example, in an image processing application requiring a pixel traversal with a constant computation on each pixel, the size of the computation (in operations) is directly proportional to the size of the image sent (the size of the task, in operations).

We can therefore express the task completion time as

$$t_i = j_i \left( \frac{1}{c_i} + \frac{R}{W} \right) + Q_i \tag{1}$$

The job completion time, $T_j$, is the length of time taken for the offload source to receive the task solutions from all offload targets. The task that takes the longest to complete is the sole determinant of the job completion time. That is,

$$T_j(\vec{j}) = \max_i t_i \tag{2}$$

where

$$\vec{j} = \begin{pmatrix} j_1 \\ j_2 \\ \vdots \\ j_n \end{pmatrix}. \tag{3}$$

Our goal is to choose the optimal assignment of tasks to minimize job completion time.

We write this problem mathematically as

$$\begin{aligned} \arg\min_{\vec{j}} T_s(\vec{j}) = \arg\min_{\vec{j}} \left( \max_i t_i \right) \\ \text{subject to } \sum_i j_i = J. \end{aligned} \tag{4}$$

The constraint

$$\sum_i j_i = J \tag{5}$$

indicates that the sum of the sizes of all sub-jobs must equal the size of the original job.

To solve this integer program, we recognize that an equivalent description of the problem is

$$\arg\min_{\vec{j}} \|\vec{t}\|_\infty \quad s.t. \sum_i j_i = J \tag{6}$$

where $\vec{t}$ is a vector of the task completion times.

It is well known in numerical optimization that a problem which minimizes an $\infty$-norm is convertable to epigraph form by introducing an auxiliary variable $y$, allowing a standard numerical solver to be used to obtain a solution. Our problem becomes

$$\arg\min_{\vec{j}} y \tag{7}$$

$$\text{subject to } y \geq |t_i| \ \forall i \tag{8}$$

$$\sum_i j_i = J. \tag{9}$$

In our case, we know that $t_i$ will be positive, so we can ignore the absolute value. To write the constraints in standard form, we move the unknowns to the left side of the inequalities:

$$\arg\min_{\vec{j}} y \tag{10}$$

$$\text{subject to } y - t_i \geq 0 \ \forall i \tag{11}$$

$$\sum_i j_i = J. \tag{12}$$

From here, we use a widely used LP-solver (lp_solve, [26]) to perform the minimization.

### B. Capturing T-HPC Utilization and Link Quality

Thus far, we have assumed that the offload source is aware of computation capability and current utilization of the available offload targets. To obtain the computation utilization from offload targets, we use an on-demand request where an offload source first floods the network with a request for utilizations. All offload targets that receive this request respond with their capacity (the number of operations per second they are capable of) and their current utilization (the length of their current task queue in seconds).

## IV. Evaluation

In our evaluation, the offload sources are mobile devices and the offload targets are much more computationally capable T-HPCs. We vary the following independent variables: task size, number of mobile devices, number of T-HPCs, and bandwidth. Since our goal is to reduce the time to complete each offload source's job, we measure *job completion time*

(JCT) in seconds as the dependent variable. We also measure the time to execute our offloading algorithm.

We evaluate our utilization-aware offloading algorithm against the following baselines:

1) **Local:** performs the entire computation locally on the offload source. It provides a fundamental baseline to demonstrate the benefit of computation offloading.
2) **Equal:** divides the job into tasks of equal size and send a task to every available target. It provides a baseline for distributed offloading on multiple targets.
3) **Architecture-aware:** decides which T-HPC to offload the entire job to based on the computational capabilities of the available targets. It distribute the job proportional to the capabilities of the devices and represents existing algorithms that consider hardware profiles of the targets when making offload decisions.

We evaluate our offloading algorithm by using a network simulator, ns-3 [27], to easily configure parameters such as number of mobile devices and T-HPCs, T-HPC capabilities, channel bandwidth, transmitted data size, etc. We conducted all experiments on a PC with a 2.8 GHz Intel Core 2 Duo processor with 4 GB of RAM. However, since ns-3 is a discrete-time network simulator, all but one of our results is independent of the specifications of the computer used to run the simulations. The only result that the computer used to perform the simulation affects is the time taken to minimize the integer program used to determine the job division, as we report the wall-clock time taken to perform this optimization.

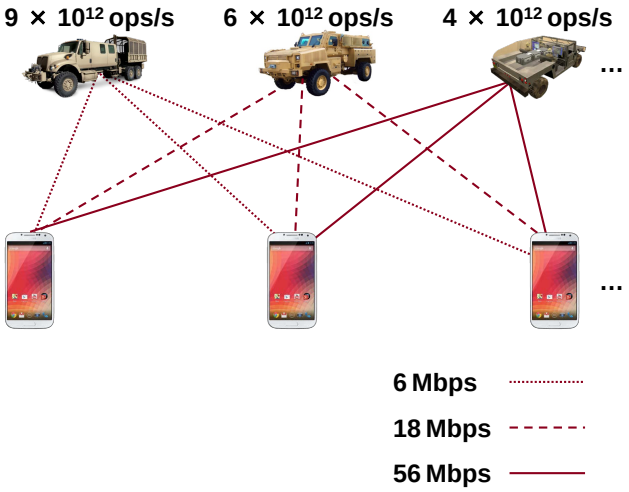


**Fig. 3.** The topology used for our evaluation modeled high-end smartphones as offload sources and three T-HPCs with varying computational capabilities as offload targets. We selected standard Orthogonal frequency-division multiplexing (OFDM) values for bandwidths and assigned them inversely to the capabilities of the offload targets

Figure 3 shows the topology modeled for the evaluation. We used the computation capability of two GeForce GTX Titan gaming supercomputer graphics cards as the high-capability offload target, and scaled it down for the medium-capability, and low-capability targets. The offload sources were modeled after current high-end smartphones such as the Samsung Galaxy S4 and Apple iPhone 5. In order to demonstrate the importance of considering the whole system state, including target utilizations and network bandwidth, we modeled a worst-case scenario in terms of network connections by assigning bandwidth inversely to the capabilities of the target. We picked standard bandwidth values used by the orthogonal frequency-division multiplexing (OFDM) method of data encoding on multiple carrier frequencies.

For every experiment, we repeated the simulation 10 times and reported the average values of JCT. In each run of the simulation, we randomly assigned the position of each device and used a propagation model in ns-3 that varies the quality of the link between nodes based on their distance to each other.

In our studies, we focus on the tactical edge scenario where the available T-HPCs are unknown ahead of job execution, but immobile. Each offload source is independent and unaware of other offload sources. They each have computation jobs that they can divide into tasks for offloading.

In our evaluation, we model a computational job as an entity requiring the execution of a certain number of operations in order to complete. We assume that the job is decomposable into tasks at operation granularity with no dependencies between the tasks. Each T-HPC processes tasks as they arrive, and queues tasks in an infinite wait list. We do not yet model priorities and task preemption, therefore T-HPCs execute tasks in a first-come first-served manner. We currently assume that the task size is proportional to the number of operations required for it to complete with the relationship between task size and operations defined by the constant $R$ in Section III-A.

## V. Results and Discussion

In this section, we present evaluations designed to demonstrate the potential of our algorithm. We evaluated the scalability of our offloading algorithm as the network size increased as well as its performance as the size of the job to be offloaded changed.

Since offloaded tasks can either be data intensive or computation intensive, we first studied the effect of these two types of jobs on system performance. The first experiment varied the transmission size from $2 \times 10^7$ B to $2 \times 10^8$ B. This allows us to evaluate the performance of offloading data intensive jobs. The second experiment varied the amount of computation required to process a job with $R = 72000$ from $2 \times 10^{12}$ Ops to $1.8 \times 10^{13}$ Ops to evaluate the performance of offloading computation intensive jobs. Figure 4 presents the performance of the utilization-aware offloading algorithm as compared to the baseline algorithms when the amount of data to be transmitted from the offload source to the offload target increases from $2 \times 10^7$ B to $2 \times 10^8$ B. As the figure shows, utilization-awareness allows the job to complete in the least amount of time for all transmission sizes evaluated.

In the scenario used for the evaluation where the offload targets had orders of magnitude more computational power than the offload sources, local execution resulted in the lowest job completion time. An interesting observation in this situation is that the more intelligent architecture-aware offloading scheme performed worse than the simpler equal distribution. This is due to the fact that in the evaluated network we set the bandwidth of links to be inversely proportional to the capabilities of the offload targets. Therefore, the architecture-aware algorithm would transmit a larger portion of the job over

lower bandwidth links than an equal distribution which causes it to perform worse than the strawman approach of equally distributing the task across targets.

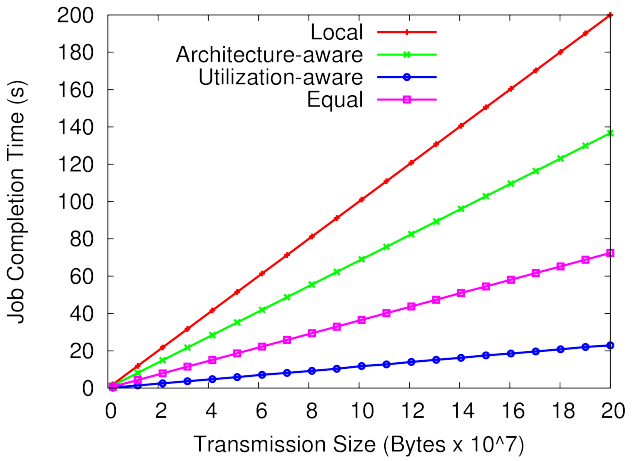


**Fig. 4.** In simulation, we observe that as the size of the job to be offloaded increases, the time to complete the job increases linearly. The job completion time for equal distribution increases at a rate of 3.6 s/B, while the utilization-aware offloading algorithm has a job completion time increase of only 1.1 s/B

Figure 5 presents the performance of the utilization-aware offloading algorithm as compared to the baseline algorithms when the amount of computation required for a job increases from $2 \times 10^{12}$ Ops to $1.8 \times 10^{13}$ Ops. In this scenario, the job completion time for equal distribution is orders of magnitude greater than the architecture-aware algorithm. The reason for this poor performance was the equal distribution across all devices causing a large amount of the computation to be executed locally. Since, the architecture-aware algorithm allocates tasks proportionally to the capabilities of the device, the amount of computation executed locally is very small resulting in a performance similar to utilization-aware offloading when the amount of data to be transmitted is small.

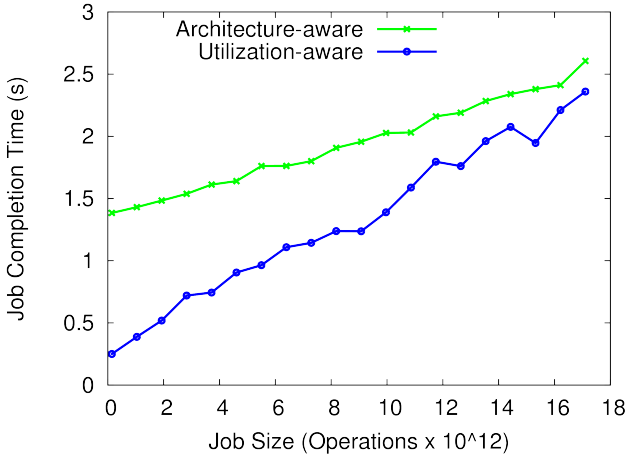


**Fig. 5.** The performance of several offloading algorithms versus the size of the computation job. The figure does not depict the performance of local execution and equal distribution because they took orders of magnitude longer to complete execution than the architecture-aware and utilization-aware algorithms

In Figure 6, we show the behavior of the system with an increasing number of offload sources. The figure shows that considering the current system state in terms of utilization and network bandwidth allows the utilization-aware offloading algorithm to scale well as the number of offload targets increase. The job completion time for the architecture-aware baseline grows steadily as the number of clients increase surpassing the equal distribution above 6 clients. As presented in Figure 4, equal distribution can outperform architecture-aware offloading as the amount of data to be offloaded increases. Thus, as more offload sources join the network, due to the architecture-aware algorithm offloading a larger proportion of its jobs over the lower bandwidth links to the faster targets, its performance degrades faster than the simpler equal distribution.

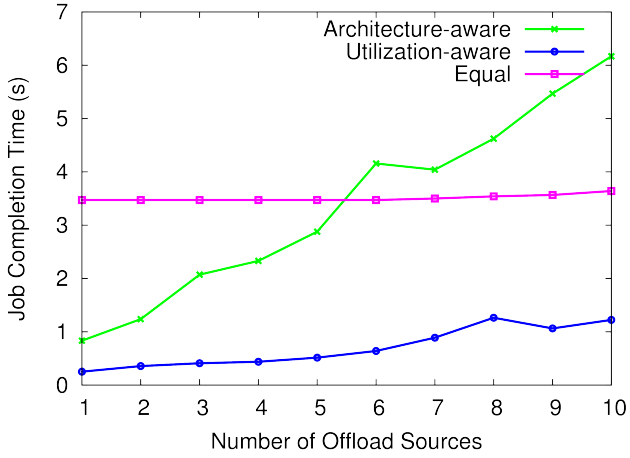


**Fig. 6.** Simulated system performance with varying numbers of offload sources. The utilization-aware algorithm outperforms the alternative algorithms

Next, we investigate how well our algorithm scales with the network size. While there are more resources available for offloading as the number of targets increase, efficiently utilizing these resources is more complicated. To study the behavior of the algorithms with more offload targets, we replicate the set of three targets seven times resulting in the number of offload targets increasing in multiples of three from three to 21, as shown in Figure 7. In this scenario, we use a job of size 2 MB with R = $1 \times 10^6$. Since architecture-aware and utilization-aware algorithms already make relatively efficient offload decisions, their performance does not improve much with the number of servers. The performance of the architecture-aware offloading algorithm approaches the utilization-aware algorithm with more servers as smaller amounts of data are offloaded over lower bandwidth links. The equal distribution sees a great benefit as the number of targets increase since the amount of computation executed locally decreases with the number of offload targets.

We note that solving the optimization problem in our algorithm does not degrade system performance as it takes several orders of magnitude less time than the jobs that we are computing.

## VI. Conclusions and Future Work

We have presented a proof-of-concept for a system that enables computation offloading by dividing jobs into tasks

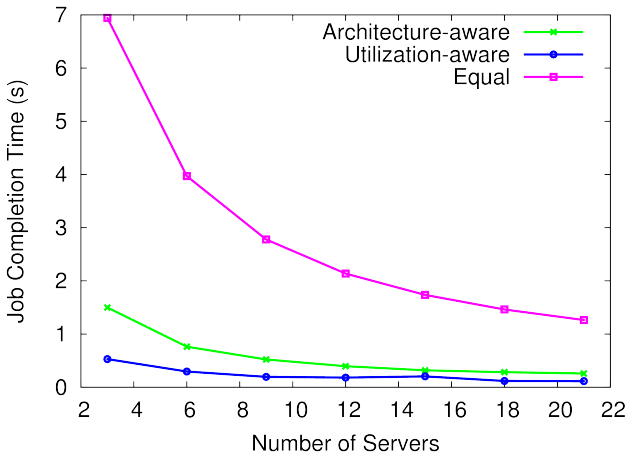


**Fig. 7.** An increasing number of offload targets allows greater parallelism as a job can be distributed across more devices

and distributing them to offload targets in a utilization-aware fashion. Using values from currently available mobile devices as offload sources and T-HPCs as offload targets, we have shown that an integer program considering bandwith, offload target utilization and architecture, and job size is able to obtain lower job completion times than architecture-aware-only algorithms. Additionally, our algorithm scales well with the number of offload sources, offload targets, and job size. We have validated our concept using several experiments performed in an ns-3 simulation.

Our work will continue by adding more capabilities to the offloading strategy and expanding the models of underlying tactical edge scenarios. We will extend the model to account for varying levels of bandwidths changing over time, including intermittent connectivity, due to mobile sources and targets, and environment conditions affecting the network strength. We will investigate extending the model to account for multiple threads on the T-HPCs and T-HPCs handling more than one task at a time. We will also account for more variables in the jobs and job division for offloading, for example different data sizes and more accurate estimates of computation time. On the tactical edge, some jobs may have higher priorities than others. Therefore, we will also extend our system to consider priority during execution.

We currently assume embarrassingly parallel jobs so that they are decomposable at the computing cycle level with no dependencies between the tasks. There are applications, such as some image processing algorithms, where this holds true, yet offloading with decomposition could apply to other kinds of applications. We plan to study how our abstract model of a job maps to applications that will executed on the tactical edge. We also plan to implement our algorithms in actual hardware, comparing the system against existing offloading implementations such as Serendipity [9] and ThinkAir [3].

## REFERENCES


[1] S. Weintraub, "Industry first: Smartphones pass pcs in sales," *Fortune*, February 2011.

[2] C. Shi, M. H. Ammar, E. W. Zegura, and M. Naik, "Computing in cirrus clouds: the challenge of intermittent connectivity," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 23–28.

[3] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 945–953.

[4] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.

[5] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 301–314.

[6] "Amazon ec2," http://aws.amazon.com/ec2/.

[7] D. Shires, B. Henz, S. Park, and J. Clarke, "Cloudlet seeding: Spatial deployment for high performance tactical clouds."

[8] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *Personal Communications, IEEE*, vol. 8, no. 4, pp. 10–17, 2001.

[9] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*. ACM, 2012, pp. 145–154.

[10] D. Kakadia, P. Saripalli, and V. Varma, "Mecca: Mobile, efficient cloud computing workload adoption framework using scheduler customization and workload migration decisions," in *Proceedings of the First International Workshop on Mobile Cloud Computing & Networking*, ser. MobileCloud '13. New York, NY, USA: ACM, 2013, pp. 41–46. [Online]. Available: http://doi.acm.org/10.1145/2492348.2492357

[11] H. Flores and S. Srirama, "Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning," in *Proceeding of the Fourth ACM Workshop on Mobile Cloud Computing and Services*, ser. MCS '13. New York, NY, USA: ACM, 2013, pp. 9–16. [Online]. Available: http://doi.acm.org/10.1145/2482981.2482984

[12] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*, ser. MCS '12. New York, NY, USA: ACM, 2012, pp. 29–36. [Online]. Available: http://doi.acm.org/10.1145/2307849.2307858

[13] R. K. Balan, M. Satyanarayanan, S. Y. Park, and T. Okoshi, "Tactics-based remote execution for mobile computing," in *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, ser. MobiSys '03. New York, NY, USA: ACM, 2003, pp. 273–286. [Online]. Available: http://doi.acm.org/10.1145/1066116.1066125

[14] T. T. J. Frey and et al., "A computation management agent for multi-institutional grids," *J. Cluster Comput.*, vol. 5, pp. 237–246, 2002.

[15] S. J. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw, "Resource management in legion," *Future Generation Computer Systems*, vol. 15, no. 5, pp. 583–594, 1999.

[16] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 59–72, 2007.

[17] L. D. Briceño, H. J. Siegel, A. A. Maciejewski, M. Oltikar, J. Brateman, J. White, J. Martin, and K. Knapp, "Heuristics for robust resource allocation of satellite weather data processing on a heterogeneous parallel system," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 11, pp. 1780–1787, 2011.

[18] L. D. Briceno, B. Khemka, H. J. Siegel, A. A. Maciejewski, C. Groër, G. Koenig, G. Okonski, and S. Poole, "Time utility functions for modeling and evaluating resource allocations in a heterogeneous computing system," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 7–19.

[19] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[20] M. J. Matarić, G. S. Sukhatme, and E. H. Østergaard, "Multi-robot

task allocation in uncertain environments," *Autonomous Robots*, vol. 14, no. 2, pp. 255–263, 2003.

[21] G. Mainland, D. C. Parkes, and M. Welsh, "Decentralized, adaptive resource allocation for sensor networks," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, vol. 2, 2005, pp. 315–328.

[22] M. Y. Arslan, I. Singh, S. Singh, H. V. Madhyastha, K. Sundaresan, and S. V. Krishnamurthy, "Computing while charging: building a distributed computing infrastructure using smartphones," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 193–204.

[23] B. D. Young, J. Apodaca, L. D. Briceño, J. Smith, S. Pasricha, A. A. Maciejewski, H. J. Siegel, B. Khemka, S. Bahirat, A. Ramirez *et al.*, "Deadline and energy constrained dynamic resource allocation in a heterogeneous computing environment," *The Journal of Supercomputing*, pp. 1–22, 2012.

[24] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 2006.

[25] H. B. Lim, M. C. Foo, Y. Zeng *et al.*, "Adaptive distributed resource allocation in wireless sensor networks," 2006.

[26] "lp_solve," http://lpsolve.sourceforge.net/5.5/.

[27] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley, "ns-3 project goals," in *Proceeding from the 2006 workshop on ns-2: the IP network simulator*. ACM, 2006, p. 13.