P^3-Mobile: Parallel Computing for Mobile Edge-Clouds

Joaquim Silva Daniel Silva Eduardo R. B. Marques Luís Lopes Fernando Silva
Faculty of Science, University of Porto & CRACS/INESC-TEC
e-mail: {joaquim.silva,edrdo,lblopes,fds}@dcc.fc.up.pt

Abstract

We address the problem of whether networks of mobile devices such as smartphones or tablets can be used to perform opportunistic, best-effort, parallel computations. We designed and implemented P^3-Mobile, a parallel programming system for edge-clouds of Android devices to test the feasibility of this idea. P^3-Mobile comes with a programming model that supports parallel computations over peer-to-peer overlays mapped onto mobile networks. The system performs automatic load-balancing by using the overlay to discover work. We present preliminary performance results for a parallel benchmark, using up to 16 devices, and discuss their implications towards future work.

Categories and Subject Descriptors D.1.3 [*Concurrent Programming*]: Parallel Programming

Keywords mobile edge-clouds, peer-to-peer networks, parallel computing

1. Introduction

The smartphone has become ubiquitous and powerful. Current estimates suggest that in 2016 more than 2 billion consumers worldwide owned a smartphone, corresponding to as much as 80% of the mobile data traffic [1, 11]. As a result, it is increasingly likely to find mobile devices in proximity of each other. The internal capabilities of current devices have also evolved substantially: multi-core processing, low-power consumption, several gigabytes of storage, a horde of specialized sensor devices, and multiple wireless communication interfaces. Likewise, wireless network technology has also evolved considerable with new standards and protocols providing for low-latency, untethered, device-to-device communication, necessary to establish higher-level network abstractions such as meshes for application use. In this context, a mobile edge-cloud is a cluster of mobile devices that provide their storage and computation resources over a communication network based on wireless protocols. It is hoped that, collectively, such an infra-structure can support game-changing locality-based applications as well as opportunistic best-effort computing.

This trend prompted many researches to investigate the computational capabilities of such edge-clouds, moving away from previous paradigms such as Mobile Cloud Computing (MCC) [3] or Cloudlets [15], in which most, if not all, computationally hard tasks were off-loaded from the devices to nearby computational nodes (as in cloudlets) or to a traditional cloud infrastructure (as in MCC). In edge-clouds [5], a computational task is spawned by some node in a mesh and a subset of the reachable nodes in the network will collaborate with processing resources to complete the task, the partial results being collected by some algorithm, centralized (e.g., sent to the spawner node) or distributed (e.g., using some distributed storage service).

Here, we address the problem of whether we can *use edge networks of mobile devices such as smartphones to perform opportunistic, best-effort, parallel computations*, as opposed to the usual paradigm of high-performance parallel computing, taking advantage of their sheer number, of data locality (low latency) and of the relatively high-bandwidth of available wireless technologies. We focus on crowd-sourcing applications like data-intensive sensing (using the manifold device sensors, camera, etc.) and sporadic computationally intensive applications, e.g., a distributed face recognition application to locate a missing person [5].

Towards this goal, we re-designed and implemented a parallel programming system developed for traditional peer-to-peer networks on top of an edge-cloud of Android devices. Parallel-Peer-to-Peer (P^3) [13], as the system was called, implemented a hierarchical peer-to-peer overlay on top of an IP network. The computation started at the root node. A dynamic work sharing strategy directs new volunteer nodes to request work from busy nodes so that they can quickly start working and contribute for the computation. P^3-Mobile [16] implements a similarly structured peer-to-peer overlay, over a WiFi network and with Android smartphones as the computing nodes. Preliminary re-

CrossCloud'17, April 23, 2017, Belgrade, Serbia.
Copyright © 2017 ACM 978-1-4503-4934-5…$15.00.
http://dx.doi.org/10.1145/3069383.3069388

sults with an embarrassingly parallel application (computing the Mandelbrot Set for the complex function family $f(z) = z^2 + c$, where $z, c \in \mathbb{C}$) show that significant speedups can be reached using up to 16 devices.

The remainder of the paper has the following structure. Section 2 overviews the literature on parallel programming systems for mobile edge-clouds. Section 3 introduces describes P^3-Mobile and describes its architecture and implementation in the Android platform. Section 4 describes the experimental setup we used to gauge the performance of the system and provides an analysis of the results. Finally, Section 5 puts forward our conclusions and thoughts for future work.

2. Related Work

Although edge-clouds are a relatively recent development in mobile computing, there have been some attempts to use these platforms to perform distributed/parallel computations [7].

Mobile Message Passing Interface (MMPI) [4] is an implementation of the MPI standard for message passing for parallel distributed computing over Bluetooth networks. MMPI builds Bluetooth meshes by interwining several piconets and provides communication and discovery services. This allows point-to-point connections to be established between any 2 devices. Parallel computations are spawned by a master device and propagated to slave devices that perform the computations and return the results.

Hyrax [9] is a port of the map-reduce [2] model for parallel computing, as implemented in Hadoop [17], to Android devices. The system uses WiFi for communication and allows applications to spawn computing jobs from a central server onto networks of mobile devices. The management of the network of available devices is performed by Hyrax internally allowing applications to abstract away from issues like churn and network topology. Hyrax provides fault-tolerance mechanisms that provide some resilience to churn.

Honeybee [6] implements an opportunistic work-sharing model that allows independent jobs to be disseminated and executed on networks of mobile devices. It uses work stealing algorithms to load-balance computational jobs among the devices in a heterogeneous network and implements some resilience mechanisms to handle churn. mCloud [10] is a proposal for an architecture in the lines of Honeybee, for which no implementation is available.

FemtoClouds [8] allows users to create networks of mobile devices, receive job submissions and schedule these jobs amongst the devices. Some central coordination is required in the form of a “control device”, responsible for the management of the network and for scheduling the jobs.

P^3-Mobile differs from the aforementioned proposals by providing a programming model that supports opportunistic parallel computations over peer-to-peer overlays mapped onto mobile networks. The system performs automatic load-balancing by using the overlay to discover work. It also implements basic resilience mechanisms that may, at least partially, mitigate the problem of churn, although we do not address this issue in this paper.

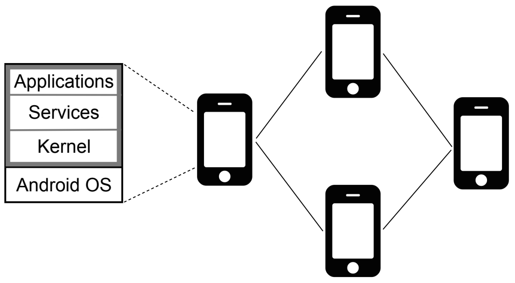

Figure 1. The P^3-Mobile architecture.

3. The P^3-Mobile system

Parallel Peer-to-Peer (P^3) is a system developed for traditional desktop/server-based networks with the goal of enabling the implementation of parallel applications on a peer-to-peer overlay [12, 13]. It provides a simple parallel programming model based on work-sharing supported by tree-structured overlays, to make node membership and discovery more efficient. In this section, we describe the main features of P^3 and its Android sibling that we call P^3-Mobile.

3.1 Overview

P^3 has the layered architecture shown in Figure 1. The kernel layer (at bottom) is the engine that drives the rest of the system, providing support for core functions such as network management and device-to-device communication. The middle layer comprises a set of generic services, including those used to handle network overlays and parallel computing, discussed in more detail below, but also for other facilities such as distributed storage. Applications on the upper layer are implemented through the use of P^3 services, e.g., an application can use the overlay and parallel processing services to distribute and perform a parallel computation, and make the computation results available through peer-to-peer messaging, as in the Mandelbrot application we use for the evaluation (Section 4), or using the distributed storage service.

The prototype P^3-Mobile is a port of the original P^3 to the Android operating system, to be used in mobile devices. The main difference is that whereas the original system used a centralized registry to implement the membership service, in P^3-Mobile devices are added to the overlay by coordinators which they discover by listening to their WiFi broadcast messages.

3.2 Overlays

P^3 overlays, illustrated in Fig. 2, are tree structures made of coordination cells. Each coordination cell (a node in the tree) groups a set of devices with bounded size (4 in the

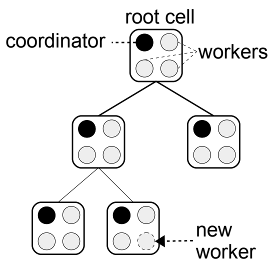

Figure 2. A P^3-Mobile hierarchical overlay.

figure). Devices in the same cell will ideally be part of the same physical network, but need not to, as traffic routing between distinct networks that support the overlay may be handled by the P^3 kernel. Each device in a cell can either play the role of a worker, merely providing resources such as processor time and disk storage, or that of a coordinator, that is in addition responsible for managing the network overlay and for supporting distributed services. For fault tolerance, there may be more than one coordinator per cell; the figure shows the simpler case where there is only one coordinator, for details see [13]. As new devices join the network, coordination cells in the overlay may eventually become full, and in that case the new devices form new descendant (children) cells leading to a tree structure; the number of descendants in a coordination cell is also bounded (2 in the figure), hence the joining procedure recurses down the tree if necessary.

Devices assume the role of coordinators as follows. The first device, the one that spawns the computation, starts a cell and is a coordinator by construction. Devices that arrive later on fill in available positions in the cell until it is full. Then, a new arrival will create a new cell and be elected its coordinator. The process continues henceforth. If a node in a cell fails or leaves, it leaves a position available in the cell that is occupied by incoming devices before creating new cells. If it is the coordinator that fails, another device in the cell is promoted to coordinator.

As the original P^3 system targeted workstation clusters, the coordinator role was preferably attributed to server machines with good capabilities in terms of Internet connection, network bandwidth, storage, etc. In P^3-Mobile, these characteristics may play a role, e.g., an Internet connection may be more adequate to connect two coordinators that are not in range of one another, while peer-to-peer communication (e.g., WiFi-direct) may suffice in the scope of a single coordination cell that aims to group nearby devices. In any case, the main appeal of P^3 is more in terms of the potential flexibility in defining overlays in a decentralized manner.

A device joins the network by looking for a root-level coordinator (there may be more than one in the root-cell) that announces its presence using WiFi broadcast. After finding such a coordinator the device uses the following the algorithm to join the overlay:

```
procedure $\mathrm{JoinOverlay}(d : \mathrm{Device}, c : \mathrm{Cell})$
    $M = \mathrm{members}(c)$
    $D = \mathrm{descendants}(c)$
    if $\neg\mathrm{isFull}(M)$ then
        $M := M \bigcup \{ d \}$
    else if $\exists c' \in D : \neg\mathrm{isFull}(c'.M)$ then
        $\mathrm{JoinOverLay}(d, c')$
    else if $\neg\mathrm{isFull}(D)$ then
        $D := D \bigcup \{ \mathrm{CreateCell}(d) \}$
    else
        $c' := \mathrm{pickCell}(D)$
        $\mathrm{JoinOverlay}(d, c')$
    end if
end procedure
```

The algorithm is recursive, and considers 4 cases: (1) if the coordination cell (c) is not yet full, the device (d) is added to it; (2) if the cell is full, but it has a descendant that is not full (c'), then the device joins the descendant cell; (3) the device forms a fresh descendant cell and becomes its coordinator, if the maximum number of descendants has not been reached; or, finally, (4) the procedure recurses to one of descendant cells. This general algorithm may be refined, especially in the choice of descendant cells (cases 2 and 4, when the algorithm recurses), to account for factors such as connectivity and current workload.

3.3 Parallel Programming Model

The P^3-Mobile parallel programming service defines an API for developers of parallel computations, which encapsulates the base logic for network overlay interactions. The overall spirit is that each device in the overlay is at any given time responsible for the execution of a portion of the workload for the entire parallel computation, subject to a workload distribution criteria. Once a portion of the workload is complete, a device seeks for further work. During the overall computation, coordination devices are responsible for keeping track of the remaining workload to execute, and trying to achieve load balance by attributing portions of the workload conveniently.

From a developer's perspective, the service is exposed by a Java class called P3ParallelTask, that defines the base logic of the service, and defines placeholder (abstract) methods to be implemented by a developer. For a parallel task, a developer must define a subclass of P3ParallelTask, providing concrete implementations of the placeholder methods. The placeholder methods and their role, illustrated in Fig. 3, are as follows:

- p3main is used to setup the computation, preliminary I/O and the initialization of data structures, before starting the computation;
- p3compute implements the algorithm used to process a unit of parallel work for the application;

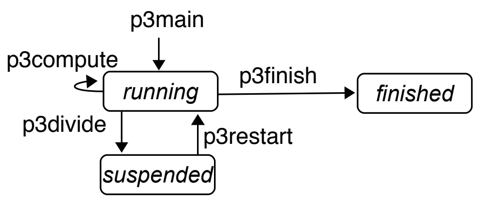

Figure 3. The P^3-Mobile programming model.

- p3divide implements the algorithm used to divide the work allocated to the current device when a work request is received from another device. For example, for most of the experiments in Section 4 (computing the Mandelbroot Set), the method simply splits the part of the complex plane assigned to the receiving device and sends half of it to the requesting device. The computation at the receiving node is also briefly suspended while this method is executed.
- p3restart is used to restart a computation in a device, either in case of failure, when a device receives new work, or after it responds to a work sharing request. Each of these cases derives or reuses a computation checkpoint, allowing the application to restart.
- p3finish is invoked when the workload attributed to a device is complete. The method is responsible for making the results of computation available, e.g., using the P^3 storage service, and freeing up any used resources.

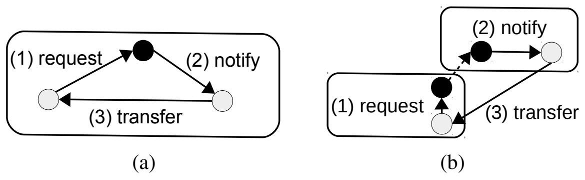

Figure 4. Work sharing in a P^3-Mobile overlay.

In support of the programming model, a number interactions occur in the background for sharing the workload among devices in the P^3 overlay. Two example interactions are illustrated in Fig. 4, where an idle worker device finds work in the same coordination cell (a) or a parent coordination cell (b). In case (a), the worker interacts with the local coordinator, which is able to find an appropriate sibling worker to transfer part of its workload; a simpler variation of this scheme (not shown) is when the coordinator transfers its own portion of the workload to the requesting worker (recall that coordination devices also perform computation). In the second case (b), there is no work to share in the local cell, hence the cell coordinator services the request by asking for work at the level of the parent cell interacting with its coordinator(s). Beyond requests by workers, note that coordinators may also, at some point, ask for further work. Similarly to the illustrated cases, they may search for work from devices in the same cell or the parent cell.

4. Evaluation

In this section we report our first evaluation of the performance of the P^3-Mobile system while running parallel applications.

4.1 Test setup

The test application computes the Mandelbrot set fractal. For a point c in the complex plan to belong to the Mandelbrot set, the sequence $z_{i+1} = z_i^2 + c$ with $z_0 = 0$ must converge. We evaluated this sequence for a 1920×1080 grid representing a subregion of the complex plane. To ensure a homogeneous computation, we performed a fixed number of 8000 iterations for each point (i.e., z_{8000} for each c), regardless of whether divergence was detected earlier in the sequence (i.e., $\exists i : |z_i| \geq 2$).

The experiments used 16 Google Nexus 9 devices running Android 7.1, each equipped with a dual-core 2.3 GHz processor, 2 GB of RAM, 16 GB of flash storage, and a Wi-Fi 802.11 card. The devices were USB-powered and used the same wireless network, enabled by an Asus RT-AC56U WiFi router. In some complementary experiments, to assess the possible benefit of peer-to-peer technologies [14], we also enabled the use of WiFi-TDLS, allowing transparent peer-to-peer links over WiFi, and of WiFi-Direct, bypassing the router entirely by letting one of the devices work as wireless access point linking up to 6 nodes (a limit imposed by Android). We did not however observe any significant differences in performance through the use of WiFi-TDLS or WiFi-Direct, as the Mandelbrot is computation-intensive rather than communication-intensive, hence we only report results for the plain WiFi router configuration.

All devices had P^3-Mobile installed from scratch and one of them spawned the computation. Data, both input arguments for tasks and output from tasks, was transferred through messages in a peer-to-peer fashion. P^3-Mobile has a simple key-value data storage system that can also be used for exchanging information between devices, but it was not used.

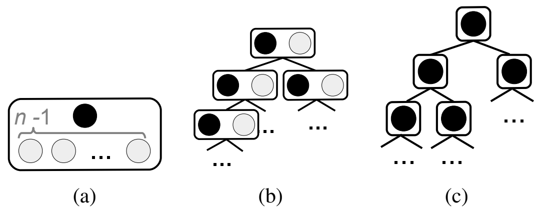

Figure 5. Overlay configurations used in the experiments.

We ran the Mandelbrot application using different overlay configurations (Fig. 5), workload division strategies and associated granularities, and varying the number of used devices (1, 2, 4, 8 and 16). To run each configuration, we employed a Python script that communicated with the devices using the `adb` Android utility. The script first performed a random selection of the devices, then activated the Mandel-

Table 1. Execution times in seconds, and speedups for the various experiments (1-device: 131.7 ± 1.5 s).

O	G	Devices			
		2	4	8	16
(a)	10	61.7±0.4 (2.1)	31.2±0.1 (4.2)	16.1±0.1 (8.2)	10.7±0.3 (12.3)
	20	59.5±0.7 (2.2)	30.0±0.3 (4.4)	16.2±0.8 (8.1)	11.5±0.6 (11.2)
	40	58.8±0.6 (2.2)	30.0±0.3 (4.4)	17.5±0.6 (7.5)	12.8±0.8 (10.3)
(a)	1	66.2±0.2 (2.0)	33.3±0.7 (4.0)	20.8±1.6 (6.3)	17.5±2.0 (7.5)
	2	66.8±0.6 (2.0)	33.2±0.5 (4.0)	21.0±1.6 (6.3)	18.3±0.7 (7.2)
	4	66.7±0.8 (2.0)	32.9±0.6 (4.0)	21.0±1.2 (6.2)	19.1±1.7 (6.9)
	8	66.6±0.4 (2.0)	34.4±0.8 (3.8)	22.3±1.1 (5.9)	20.4±1.4 (6.5)
(b)	1	66.2±0.2 (2.0)	35.2±0.4 (3.7)	24.3±0.3 (5.4)	23.5±1.2 (5.6)
	2	66.8±0.7 (2.0)	35.8±0.6 (3.7)	25.3±0.8 (5.2)	25.6±1.2 (5.1)
	4	66.7±0.8 (2.0)	36.2±0.3 (3.6)	25.4±1.5 (5.2)	22.3±1.7 (5.8)
	8	66.6±0.4 (2.0)	36.8±0.3 (3.6)	25.4±1.0 (5.2)	23.4±1.4 (5.6)
(c)	1	66.2±0.3 (2.0)	35.3±0.8 (3.7)	23.8±2.5 (5.5)	18.6±1.5 (7.1)
	2	66.4±0.4 (2.0)	34.9±0.5 (3.8)	24.6±3.0 (5.4)	19.0±2.0 (6.9)
	4	66.9±0.7 (1.9)	36.3±0.6 (3.6)	24.0±3.0 (5.5)	17.8±1.2 (7.4)
	8	66.7±0.8 (1.9)	35.8±0.8 (3.7)	24.0±3.0 (5.5)	19.8±1.4 (6.6)
(c)	1	66.0±0.2 (2.0)	34.7±1.0 (3.8)	23.6±2.9 (5.6)	15.9±3.3 (8.3)
	2	66.1±0.6 (2.0)	34.4±0.6 (3.8)	23.6±3.5 (5.6)	15.2±1.8 (8.6)
	4	66.1±0.5 (2.0)	33.5±1.6 (3.9)	22.9±3.4 (5.7)	16.4±2.6 (8.0)
	8	67.0±0.4 (2.0)	34.4±1.0 (3.8)	24.5±2.3 (5.4)	16.6±2.4 (7.9)

brot application in these devices simultaneously. We measured the average execution time of 5 runs and the corresponding 95 % Gaussian distribution confidence interval, listed in Table 1 along with the average speedup over the 1-device configuration that took 131.7 ± 1.5 seconds to execute.

In this table, O stands for the overlay configuration, as described in Figure 5, and G is the granularity of the tasks used in the runs. The first 4 blocks of rows correspond to the speedups given in Figures 6 to 9; the average load balance per device for the best-case granularity is also depicted in these figures (at bottom in each figure). The fifth block gives the speedups for overlay configuration (c) measured without including the time spent in overlay formation, as discussed later in the text.

4.2 Analysis

The best-case configurations correspond to using overlay (a) in Fig. 5, in which n devices form a single-cell overlay with 1 coordinator, and task division proceeds by attributing a fixed number of lines in the z-plane, the task granularity, to each device upon a work request. Operationally, the overlay works as a master-slave computation as $n-1$ devices ask the coordinator for work. The speedups are good (Fig. 6, top): linear up to 8, and up to approximately 13 for 16 devices; the 95% confidence intervals for the speedups are not represented for clarity but are at most 2% of the values in the worst case. Some slightly super-linear speedups were observed for the configuration (a) that may be originated by a number of conditions that are hard to identify precisely, e.g., network conditions, memory access patterns.

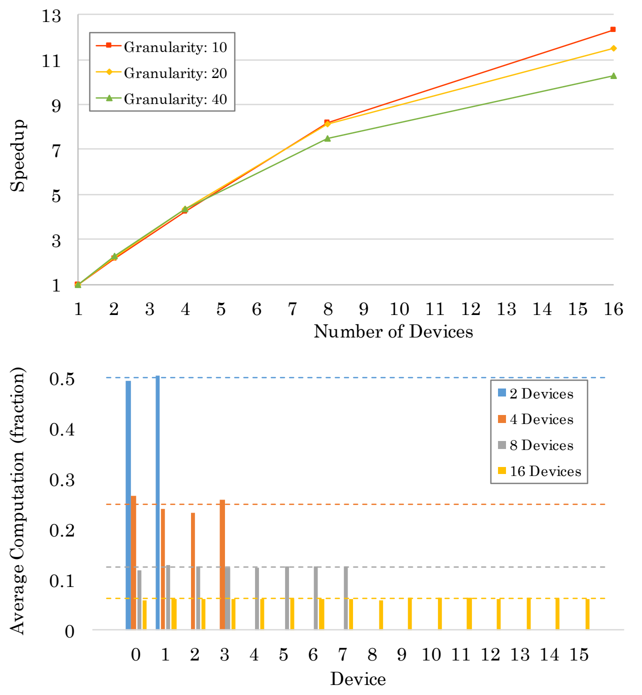

Figure 6. Speedup and load balancing for configuration (a), using fixed task size division.

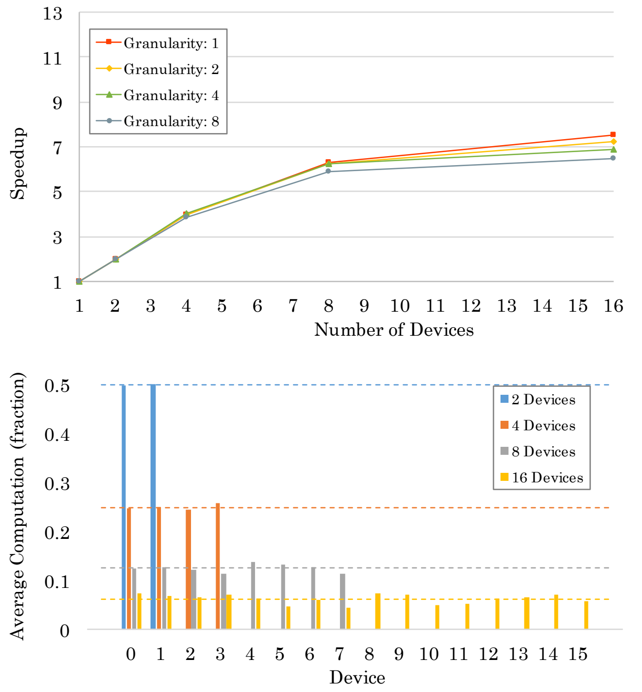

Figure 7. Speedup and load balancing for configuration (a), using halving work task strategy.

The good load-balance achieved (Fig. 6, bottom) partially explains these results, along with adequate choices for task granularity; the dashed lines in these graphics represent the ideal work fraction, $1/n$, for each device in a collection of n devices (in these experiments $1/2$, $1/4$, $1/8$ and $1/16$). The performance degradation for 16 devices can be tracked down to the underlying star topology of the WiFi network, and the overlay formation time which in the worst case can take up to 2 seconds (18% of the total execution time).

Fig. 7 shows results again for the single-cell overlay, but considering a different workload sharing scheme. In this case, each device that requests a task from another device computing k lines will receive $k/2$ lines to compute, i.e., work is divided in half. The granularity is defined as the smallest task size for which a division is allowed. The intuition for this scheme, originally used in the P^3 system, is to emulate the growth of the overlay tree and push big chunks of work down the tree; we also use this scheme for hierarchical overlay configurations discussed below. Speedups (Fig. 7, top) grow almost as good as in the previous workload sharing scheme up to 8 devices, but then have a modest increase up to 16 devices (about 7.5). Observing the workload distribution (Fig. 7, bottom) it is clear that adequate load-balancing was not achieved for higher number of devices with impact on the speedup.

Fig. 8 shows the results for overlay (b) in Fig. 5. In this case, the overlay is a binary tree where each cell is formed by one coordinator and one worker. Speedups (Fig. 8, top) are worse than the above examples for the single-cell overlay, almost 6 with 16 devices. An analysis of the logs showed that there where a number of factors contributing to these modest results. First, as above, the work sharing algorithm did not provide for a balanced work load at runtime (Fig. 7, bottom). This, however, is not a problem of the algorithm alone. In fact, overlay formation in this configuration is 4 times longer than the previous examples which contributes to an asymmetric work load. The initial network formation corresponded to 35% of the total run time. And, finally, we have contention from the access point due to increased communication.

Finally, Fig. 9 shows the results for overlay (c) in Fig. 5, a binary tree as in case (b), but where each cell is formed by only one (coordinator) device. Speedups (Fig. 9, top) are worse than those of configuration (a) with balanced tasks (Fig. 6) but are comparable to those obtained for the same configuration with the “divide by 2” work sharing algorithm (Fig. 7), especially for 16 devices. This is an improvement relative to the previous configuration and is connected with the fact that more balanced tree overlays can be formed with the (c) configuration than in (b). The speedups however never get close to the best case scenario as in this configuration the initial network setup corresponds to 50% of the total run time, so that the last devices to join in, usually, have little work left to perform.

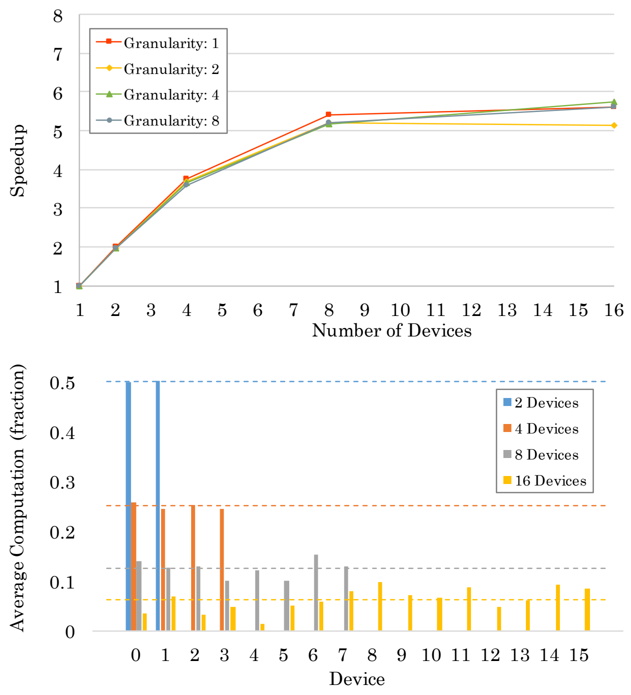

Figure 8. Speedup and load balancing for configuration (b).

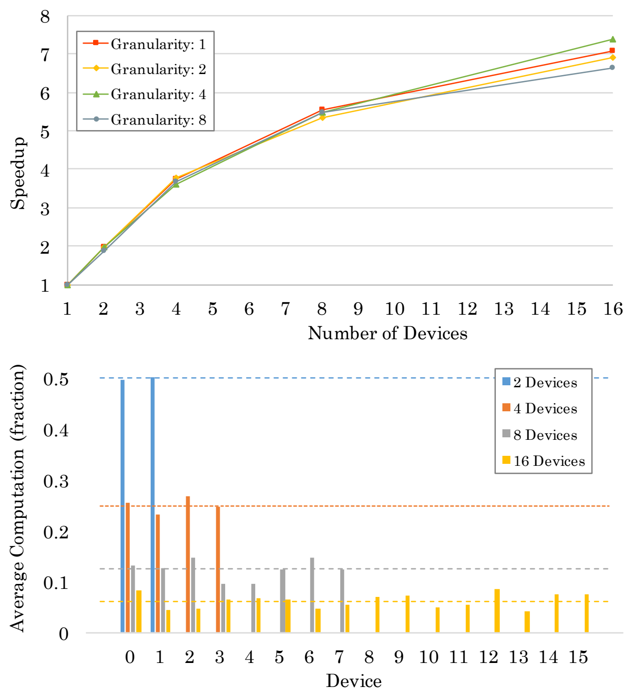

Figure 9. Speedup and load balancing for configuration (c).

To understand the impact of the overlay formation overhead in these results we ran the same experiments for all the configurations but this time we implemented a barrier that starts the computation only once all nodes have joined the overlay. The results for configuration (c), shown in the last block of Table 1, highlight that some improvement is obtained (a maximum speedup of 8.6 versus 7.4 for 16 devices) but it is clear that the work-sharing strategy is still a limiting factor.

5. Discussion and Conclusion

The results presented in this paper show that P^3-Mobile offers substantial speedups for simple, embarrassingly parallel applications, such as the Mandelbrot. They also show that the choice of overlay and work sharing strategy impacts performance, as evidenced by the fact that the speedups deviate from linear as the number of devices grows.

More work is clearly required on the system to support other overlays and, especially, more adaptive and intelligent work sharing strategies, to improve performance, with the added benefit of providing some level of fault-tolerance in the presence of churn. In particular, scheduling should be sensitive to differences in the computational capabilities of the devices and to the topology of the overlay. The construction of the overlays, e.g., the assignment of coordinator roles, should take into consideration the capabilities of the devices as well.

Another aspect of this work that was not addressed in this paper is the energy issue: what is the impact of the parallel computations and data exchange between devices in their batteries? We have reasons to believe that the impact of communication will not be very significant for applications that are broken into mostly independent tasks [14]. The energy impact from local computation, on the other hand, may be quite substantial for Mandelbrot and other applications like face recognition, but the latter would be executed only in sporadic situations.

Acknowledgments

We would like to thank the anonymous reviewers, and the invaluable help of our colleague João Rodrigues in setting up the experiments. This work has been sponsored by projects HYRAX (CMUP-ERI/FIA/0048/2013), funded by FCT, and SMILES (NORTE-01-0145-FEDER-000020), funded by NORTE 2020, under PORTUGAL 2020, and through the ERDF fund.

References

[1] Cisco. Cisco visual networking index: Global mobile data traffic forecast update, 2015–2020 white paper, 2015. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html`.

[2] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1): 107–113, 2008.

[3] H. T. Dinh, C. Lee, D. Niyato, and P. Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[4] D. C. Doolan, S. Tabirca, and L. T. Yang. Mmpi a message passing interface for the mobile environment. In *Proc. MoMM'08*, pages 317–321. ACM, 2008.

[5] U. Drolia, R. Martins, J. Tan, A. Chheda, M. Sanghavi, R. Gandhi, and P. Narasimhan. The case for mobile edge-clouds. In *Proc. UIC'13*, pages 209–215. IEEE, 2013.

[6] N. Fernando, S. W. Loke, and W. Rahayu. Honeybee: A programming framework for mobile crowd computing. In *Proc. MOBIQUITOUS'12*, pages 224–236. Springer, 2012.

[7] N. Fernando, S. W. Loke, and W. Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84 – 106, 2013.

[8] K. Habak, M. Ammar, K. A. Harras, and E. Zegura. Femto clouds: Leveraging mobile devices to provide cloud service at the edge. In *Proc. CLOUD'15*, pages 9–16. IEEE, 2015.

[9] E. E. Marinelli. Hyrax: Cloud computing on mobile devices using mapreduce. Master's thesis, Master's Thesis, Carnegie Mellon University, 2009.

[10] E. Miluzzo, R. Cáceres, and Y. F. Chen. Vision: Mclouds - computing on clouds of mobile devices. In *Proc. MCS'12*, pages 9–14. ACM, 2012.

[11] MobiForge. Global mobile statistics 2016, q2 report. `http://mobiforge.com/`, 2016. last visited in 05/09/2016.

[12] L. Oliveira. P^3: Parallel peer-to-peer. Master's thesis, Computer Science Department, University of Porto, 2003.

[13] L. Oliveira, L. Lopes, and F. Silva. P^3 (Parallel Peer-to-Peer): an Internet Parallel Programming Environment. In *Workshop on Web Engineering and Peer-to-Peer Computing*, pages 274–288. Springer, LNCS 2376, 2002.

[14] J. Rodrigues, J. Silva, R. Martins, L. Lopes, U. Drolia, P. Narasimhan, and F. Silva. Benchmarking wireless protocols for feasibility in supporting crowdsourced mobile computing. In *Proc. DAIS'16*, pages 96–108. Springer, 2016.

[15] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for VM-based cloudlets in mobile computing. *Pervasive Computing*, 8(4):14–23, 2009.

[16] D. Silva. P3 Mobile: Parallel Peer-to-Peer Computing on Mobile Devices. Master's thesis, Computer Science Department, University of Porto, 2016.

[17] T. White. *Hadoop: The definitive guide*. O'Reilly, 2012.