PowerShare:

A Distributed Processing System for Android Devices

Vignesh Pradeepkumar, Shubham Jayawant, Khoshrav Doctor, Prof. Sanjay Vidhani

Department of Information Technology
K. J. Somaiya College of Engineering

Abstract*—— There has been a significant increase in the number of mobile devices sold and their computation capabilities in recent years. The computational capability of a cluster of mobile devices presents us with an opportunity to utilize those resources in a number of real world applications. We have developed PowerShare- an Android based application that uses a combination of Wi-Fi-P2P and Bluetooth to form an infrastructure with nearby devices. The network represents a shared resource pool of all the connected devices. By taking into account the load each node can handle, PowerShare delegates tasks optimally to every connected node in the network. PowerShare's optimal allocation function coupled with the unutilized computational resource of a mobile device can be used in many distributed processing applications. The design and experimental results based on a prototype are presented. The results show that PowerShare successfully optimizes task allocation to nodes and utilizes their computing resources efficiently.

***Index Terms*—— Android, Bluetooth, Distributed Computing, Distributed Processing, PowerShare, Wi-Fi, Wi-Fi-P2P.**

I. INTRODUCTION

Mobile devices have become everyday utilities. Around 2.1 billion units were shipped in 2015.Around 84%[1] of the 349 million smartphones shipped in the first quarter of 2016 were based on an Android OS. The spectrum of Android based smartphones range from the high-end Samsung Galaxy S6 which came with 3GB RAM and a 2.1 GHz Quad-Core Cortex A-57 processor to the low end Samsung Galaxy V which came with 512 MB RAM and a dual core 1.2 GHz processor. The resources present in these mobile devices are rarely utilized to their full capacity. Everyday users rarely utilize the computing resources in their smartphones to their maximum limit. The resources, however, if utilized to their extent can be used in a number of situations that would otherwise require a complete distributed processing infrastructure. For example, a white-hat tester might be able to form a network and implement a Brute-force attack using a cluster of mobile devices by delegating each node with tasks. The attack will be implemented faster as the processing time of the entire task will be subdivided by the number of nodes and each node will process the data it receives in parallel. In this paper we present PowerShare, an Android application that provides the functionality of forming a hybrid network of mobile devices for distributed computing using Wi-Fi-P2P(for data transmission) and Bluetooth(for control and synchronization).The application can be used for a number of tasks that can be optimized in a distributed environment. Tasks such as web-scraping can be seamlessly distributed among available devices resulting in distribution of the computing as well as the bandwidth load required for such a task.

In the following sections, Section 2 states past work related to the field of distributed computing in mobile devices using Wi-Fi and Bluetooth. Section 3 gives information regarding the protocols and tools used in the implementation. Section 4 describes the types of tests conducted using the application and the results obtained from the same. In Section 5 conclusions and future scope of the application are detailed.

II. Related Work

Implementations of distributed computing based on ad-hoc networks on smartphones are relatively new and research is still being done to make the system scalable and reliable at the same time.

WDC on Android Platform describes a method to interconnect mobile devices over a wireless mesh network and distribute tasks to optimally calculated number of nodes using Wi-Fi architecture[2]. However this does not leverage the significantly more powerful and secure[3] Wi-Fi Direct standard. Hinojos et al. describe BlueHoc[4]which advanced on existing architectures based on Bluetooth-based distributed processing in smartphones. Significant advances in the Bluetooth protocol enable us to build a more robust system. PowerShare uses dynamic splitting. Instead of the traditional procedure of creating equal splits for each device, PowerShare assigns tasks to a device in proportion to it's capabilities. Issues regarding device overloading(assigning more data than a device can handle)can be avoided by this.

DroidCluster[5] provides a way to connect mobile-devices using Wi-Fi that can be used for distributed computing. However due to emergence of much faster albeit location-restricted technologies such as Wi-Fi-Direct , we can build a system that is both fast, robust and dynamic in comparison to earlier work in the same field.

III. Architecture

A. *Wi-Fi Direct and Bluetooth Technology*

Wi-Fi Direct is implemented using a software access point or Soft AP embedded in supporting devices. This allows devices in the vicinity of the Soft AP to discover it and establish connection easily. Wi-Fi Direct allows transfer speeds of up to 250Mbps and supports distances of more than 600 feet. The standard uses WPA2 security using AES 256-bit encryption. Wi-Fi Direct is compatible with Android Devices above API level 14.

Bluetooth 4.0 uses the 802.11 networking standard and is capable of reaching speeds of up to 25Mbps and reach distances up to 200 feet. Security is provided using AES 128-bit Encryption[6].

B. *Processing in Android*

PowerShare was developed using Android SDK, as opposed to Android NDK which supports native languages such as C and C++. SDK is implemented using Java which is the core language of Android.

Android uses its own version of JVM called Dalvik until Android 5.0 after which ART(Android Runtime) became the default JVM for Android. ART brought significant improvements to processing times as it compiles the dalvik bytecode into native machine code on installation using AOT(Ahead-of-Time) compilation as opposed to dalvik which used JIT(Just-in-Time) compiler which executes each time the app is run. Since processing times are directly related to power consumption, ART ends up reducing power consumption as well. Improvements to ART in the form of the Optimizer Compiler made the Android JVM much more efficient in terms of speed and power consumption. Tests based on Heavy mathematical functions for over a million iterations[7] proved that in 64-bit Android 6.0 device, Java code ran about 7% faster than C based program for the same function. Keeping this and future development of Android JVM in mind PowerShare was built using SDK.

C. *PowerShare Architecture*

PowerShare system architecture is based on Client-Server system where clients connect to the server using Bluetooth first and Wi-Fi-Direct afterwards. This multifold system is mainly used to separate the control and data transmission modules. The system is set up to exploit the advantages of both the Bluetooth and Wi-Fi Direct standards.

Bluetooth allows for a maximum of 8 simultaneous connections, but the device can be setup in such a way it can act as a server and a client at the same time. This type of network where devices act as client and servers simultaneously is mainly used to form scatternets[8] which are made up of overlapping smaller networks called piconets. Wi-Fi Direct itself supports one to many connections but since PowerShare uses a combination of Bluetooth and Wi-Fi Direct, the current prototype is currently limited to 8 devices. By delegating the control and synchronization tasks to the Bluetooth standard we can use the faster and secure Wi-Fi Direct to handle larger data transmission between nodes.

The current system of PowerShare involves decided client Android devices sending requests to a main central server device via the Bluetooth protocol to

form a Bluetooth network. The server node informs the clients the port using which communication with respect to data transfer is to be conducted. Once a Bluetooth network is established, the Server then sends connection requests to all the client nodes to form the secondary Wi-Fi Direct network. The result of these two procedures is the formation of a hybrid network as shown in Figure 1 where the Client-Server system is realized using both the Bluetooth and Wi-Fi P2P protocols simultaneously.

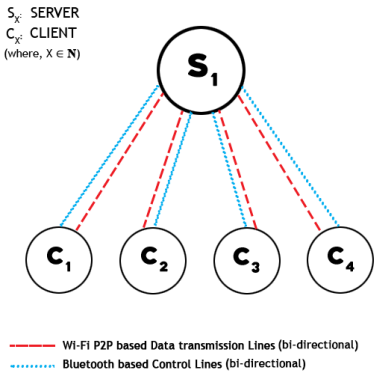


Figure 1: PowerShare architecture

Once the hybrid network is successfully setup, the client nodes send their system snapshots consisting of information regarding their API level, available RAM, available Processing Power, Battery Power left and Internet Connection strength to the Server node via Bluetooth. Apart from this information, the client nodes also send ping messages to the server after time intervals of 3 seconds. These are required to determine a client's current status. If a client fails to send a ping alert, then the server assumes a connection breakdown. The Server maintains a record of all of its connected clients containing the client's unique identifier, the aforementioned snapshot information sent by the client and client's current status. The overall score of the device is calculated by incorporating weights given to a particular resource based on its importance by the user setting up the system. The weights are mentioned in the algorithm to be distributed itself. The server then aggregates the snapshot details from all the devices and assigns scores to the connected clients based on this and decides the percentage of task each client should receive. This idea is based on the fact that each device is a non-similar and unique entity in comparison to the other devices on the network. Instead of delegating tasks equally to devices, optimally distributing them according to the current state and overall capacity of the device ensures no device is delegated a task it cannot handle. Once the delegation is calculated the server sends the calculated device-specific data to the client using Wi-Fi P2P.The device receives the data and performs the specified algorithmic function on it. The obtained results are sent back to the server by all the connected clients using Wi-Fi Direct. The server aggregates the results as they arrive. Node failure is also handled in the system. Since the server is always aware of every client's status, connection breakdowns are easy to identify. Once the server detects a client has failed, it retransmits the data delegated to the failed client to a connected and active client who would be able to take over the role of the downed client.

IV. IMPLEMENTATION AND TESTING

A. *Algorithm*

Algorithm to be implemented on the distributed system a(k), weight constants per resource of that algorithm $battery\_level_{a(k)}$, $ram_{a(k)}$, $net_{a(k)}$, $free\_cpu_{a(k)}$, $api\_level_{a(k)}$, battery level of *i*th slave node $battery\_level_i$, RAM details of *i*th slave node device $free\_ram_i$, internet access availability of *i*th slave $node\_net_i$, CPU availability of *i*th slave node $free\_cpu_i$, where ,1 <= *i* <= no of connected slave nodes

Steps:

1) i ← 1

2) TOTAL WEIGHT ← 0

3) Sort connected clients in ascending order of their API level

4) api_level_multiplier ← $api\_level_{a(k)}$

5) If i ≤ no of connected client nodes go to 6 Else go to 11

*6) Calculate total score of ith device as $total\_weight_i$ = ($battery\_level_{a(k)}$ * $battery\_level_i$) + ($ram_{a(k)}$ * ram_i) + ($free\_cpu_{a(k)}$ * $free\_cpu_i$) + ($net_{a(k)}$ * net_i)+(api_level_i * api_level_multiplier)*

7) TOTAL_WEIGHT ← TOTAL_WEIGHT + $total\_weight_i$

8) i ← i+1

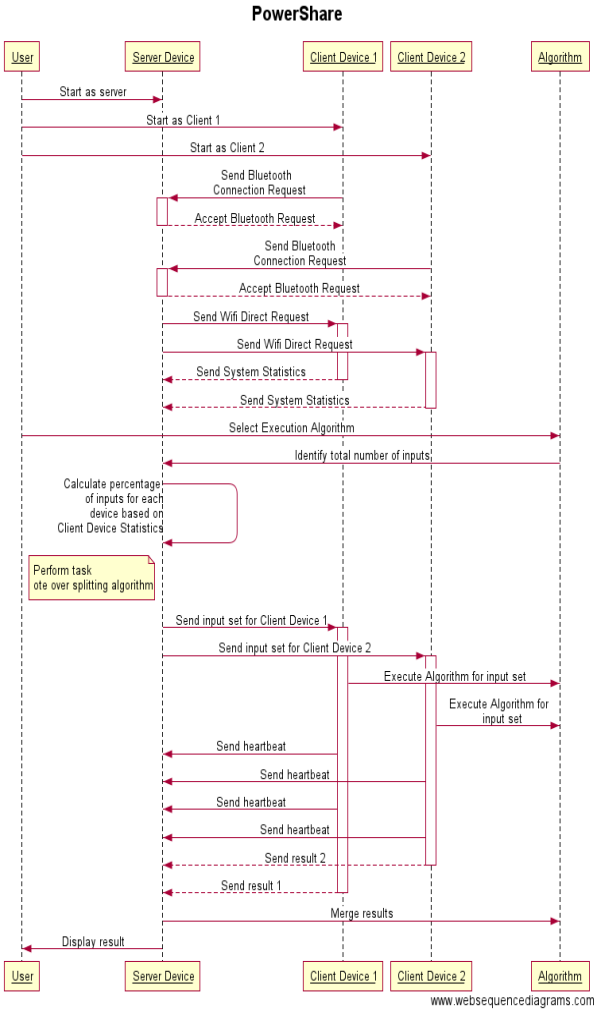
9) If api_level_i < api_level_{i-1} and api_level_multiplier is not equal to 1 then, api_level_multiplier ← api_level_multiplier -1

10) Go to 5

11) i ← 1

12) If i ≤ no of connected client nodes go to 13 Else go to 16

*13) Caclulate size of data to be assigned to ith device as no of inputs$_i$ = (total_weight$_i$ / TOTAL WEIGHT) * total no of inputs*
14) i ← i+1
15) Go to 12
16) Generate input list(i)←no of inputs$_i$ number of inputs of algorithm a(k) and send it to client$_i$
17) Execute a(k) on client$_i$ for input_list(i)
18) Generate result$_i$ and send it back to server
19) i ← 1, Result ← null
20) If i ≤ no of connected client nodes go to 21 Else go to 24
21) Result ← merge(Result, result$_i$)
22) i ← i+1
23) Go to 20
24) Stop



B. *Testing*

The tests were conducted in a 1 Server - 4 Clients environment. The devices are specified in Table 1.

Table 1:Specifications and Roles of Android Devices used for testing

Name	RAM	CPU	API	Role
Samsung Note 4 Edge	3 GB	Quad-core 2.7 GHz Krait 450 (Snapdragon 805)	19	**Server**
Samsung Grand Quattro	1GB	Quad-core 1.2 GHz Cortex-A5	16	**Client**
Samsung Galaxy S4	2 GB	Quad-core 1.6 GHz Cortex-A15	21	**Client**
Samsung Galaxy S5	2 GB	Quad-core 2.5 GHz Krait 400	23	**Client**
Samsung Galaxy Grand	1 GB	Dual-core 1.2 GHz Cortex-A9	19	**Client**

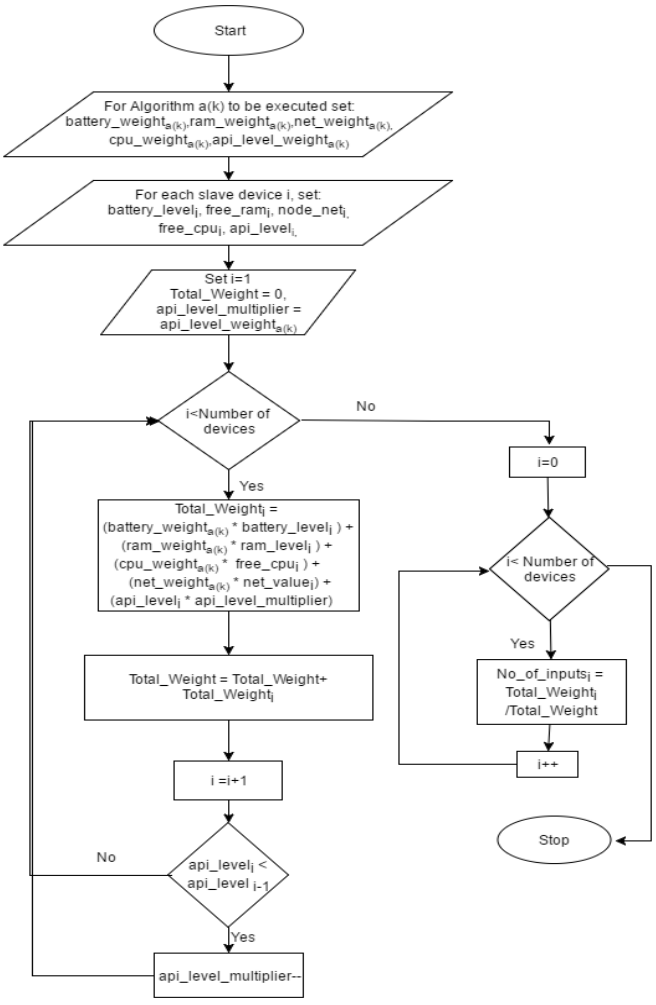


Figure 2 Splitting Algorithm

Algorithm 1

The first test algorithm is a simple program which finds the number of times a number occurs in a file containing 5 million numbers, with values between 0 and 9,99,999. The data was specifically generated for testing and was generated by using the Random.nextInt() Java function to get a uniformly distributed set of values. The test data is fed to the server. The server follows the connection protocol and prompts the user to select the test data file. After this the data is split into multiple parts based on the score of each client. The data is transferred via Wi-Fi Direct and the client processes the received input and sends the occurrence of the number back to the server via another text file. The server merges all the received inputs and presents the final output to the user. The time shown in the data below is inclusive of the time taken by the server to split the file and to aggregate the results. Tests were conducted on input data of size 1 million to 5 million. The results obtained are shown in Table 2 and represented graphically in Figure 3.

Table 2 Computing times for Algorithm 1 with 4 clients

Number of Rows (in Millions)	Time (in Seconds)
1	4.023
2	8.421
3	13.434
4	27.129
5	53.701

The same task was also run by varying the number of clients in the network. A data source of 5 million numbers was used and the number of clients were gradually increased. The time taken in running the algorithm shows a drastic improvement as more devices are added proving that the system is efficient. The results obtained are shown in Table 3 and represented graphically in Figure 4

Table 3: Computing time for Algorithm 1 with varying number of Devices

Number of Devices	Time (in seconds)
1	230.898
2	129.405
3	88.149
4	54.112

Algorithm 2

The second application was a net-intensive task designed to test the efficiency of the system in sharing internet bandwidth and overall processing

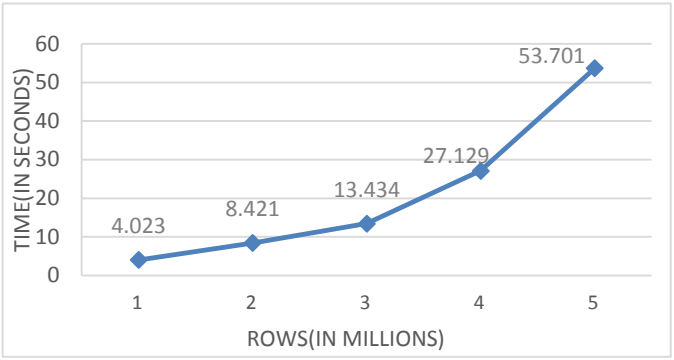


Figure 3: Graphical Representation of execution times for Algorithm 1 using 4 clients

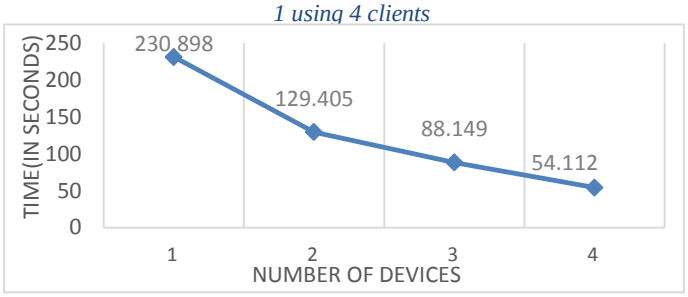


Figure 4: Graphical Representation of execution times for Algorithm 1 using varying number of clients

load. The task was to scrape web-URLs from Google News on specified topics and aggregate all the obtained results. To perform this test, a text file was generated with 1250 rows of topics. The client receives a part of the file based on its computing power and network connection strength. For a network based task the score of a client was calculated keeping its network connectivity in consideration. Jsoup was used to parse HTML data and collect the links. The client searches for a topic using a modified URL for Google News and collects the top links from the first page of the result and writes them into a file. The resultant text files are then aggregated at the server. The task was performed on a network connection with speed of 1Mbps.As with the earlier application time obtained is inclusive of the time taken by the server to split the file and aggregate the results. The results obtained are shown in Table 4. A graphical representation of the same are shown in Figure 5. The same task was also run by varying the number of clients in the network. The time taken in running the algorithm once again highlights the positives of PowerShare. The results obtained are shown in Table 5 and represented graphically in Figure 6.

Table 4: Computing times for Algorithm 2

Number of Topics	Time (in Seconds)

250	56.369
500	115.291
750	186.349
1000	220.629
1250	276.444

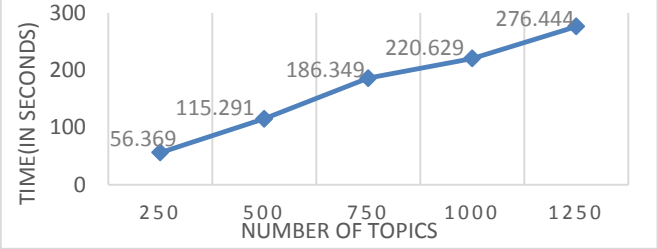


Figure 5:Graphical Representation of execution time for Algorithm 2

Table 5: Execution time for Algorithm 2 with varying number of devices

Number of Devices	Time (in seconds)
1	1118.348
2	570.0
3	406.008
4	275.812

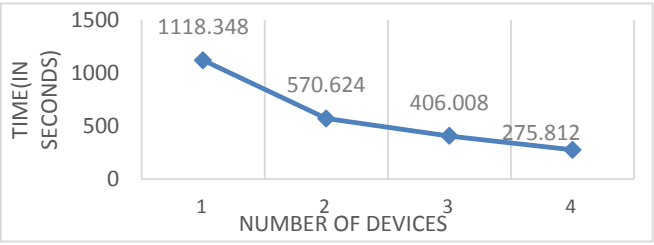


Figure 6: Graphical representation of computing time for Algorithm 2 with 1250 inputs and varying devices

V. Conclusion

Based on the tests conducted it can clearly be observed that the using PowerShare a stable network can be setup and be used in various computing applications. The short setup time and computing time show the potential for a Hybrid network based on Bluetooth and Wi-Fi direct in the field of distributed computing using smartphones.

The computing power in mobile devices is only going to increase, applications should thus be developed that are able to aggregate and harness these capabilities. PowerShare is still in its nascent stage with huge scope for advancements. The applications of a fast, reliable and easy to set-up mobile based distributed network range from basic number crunching to computing-intensive sensor networks. Future scope may involve creating a services that require real-time response and quick setup such as traffic information processing or geographical information and real-time weather processing. The simplicity of PowerShare lies in its architecture that uses basic in-built protocols such as Bluetooth and Wi-Fi P2P found in all the modern-day smartphones. Developments are aimed at incorporating all the other mobile Operating Systems such as iOS and support for Windows- based smartphones. Improvements in efficiency can be obtained by implementing runtime load distribution. and by incorporating popular distributed system concepts such as secondary root node for handling root node failures.

VI. References

[1] Gartner (May 2016) *"Gartner Says Worldwide Smartphone Sales Grew 3.9 Percent in First Quarter of 2016,"* http://www.gartner.com/newsroom/id/3323017, 2016

[2]Kiran Karra, *"Wireless Distributed Computing on the Android Platform,"* https://theses.lib.vt.edu/theses/available/etd-10012012-214953/unrestricted/Karra_K_T_2012.pdf ,2012

[3]"Discovering the Upcoming Wi-Fi Direct Standard," http://www.ciscopress.com/articles/article.asp?p=1620205&seqNum=2,2010

[4] G. Hinojos, C. Tade, S. Park, D. Shires, and D. Bruno *"BlueHoc: Bluetooth Ad-Hoc Network Android Distributed Computing "* http://worldcomp-proceedings.com/proc/p2013/PDP2883.pdf,2013

[5] Felix Büsching, Sebastian Schildt, Lars Wolf, *"DroidCluster: Towards Smartphone Cluster Computing -- The Streets are Paved with Potential Computer Clusters"*2012 32nd International Conference on Distributed Computing Systems Workshops,2012

[6] "Wi-Fi Direct vs. Bluetooth 4.0: A Battle for Supremacy," http://www.pcworld.com/article/208778/Wi_Fi_Direct_vs_Bluetooth_4_0_A_Battle_for_Supremacy.html , 2010

[7] Gary Sims *"Java vs C app performance – Gary explains"* http://www.androidauthority.com/java-vs-c-app-performance-689081/ , 2016

[8] R. Shepherd, J. Story, and S. Mansoor, "Parallel computation in mobile systems using bluetooth scatternets and java," in *Parallel and Distributed Computing and Networks*, 2004, pp. 159–164.f[7