UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Development of grid computing middleware for Android smartphones

Author:
Volodymyr CHERNETSKYI

Supervisor:
Oleg FARENYUK

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2021

Declaration of Authorship

I, Volodymyr CHERNETSKYI, declare that this thesis titled, "Development of grid computing middleware for Android smartphones" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"In life, nothing good comes out of hurrying."

Shikamaru Nara

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences
Department of Computer Sciences

Bachelor of Science

Development of grid computing middleware for Android smartphones

by Volodymyr CHERNETSKYI

Abstract

Educational institutes that work with computational science problems need computing power. Volunteer computing is a trend of donating computer resources by forming a distributed system. This thesis aims to develop middleware for the grid of Android smartphones. In this context, a computer grid is defined as the distributed computing system formed of geographically dispersed heterogeneous computers connected by a global network. Bidirectional client-server communication is proposed to shift from the client pulling the work unit to the server pushing it. As a result, a grid computing system for Android smartphones is designed considering the specifics of the Android operating system.

Acknowledgements

I want to thank my classmate Yulianna Tymchenko for helping me derive this thesis's subject.

Big thanks to my supervisor Oleg Farenyuk for mentoring the development of the thesis on all its stages.

I am grateful to my classmates Andrii Koval, Hermann Yavorskyi, and Mykola Biliaiev for the care we showed to the progress every one of us made.

Lastly, I want to thank my best friend Ostap Viniavskyi for all the help and support he provided during the time we know each other.

Contents

List of Figures

List of Tables

List of Abbreviations

ART	**A**ndroid **R**un**t**ime
BOINC	**B**erkeley **O**pen **I**nfrastructure for **N**etwork **C**omputing
CPU	**C**entral **P**rocessing **U**nit
FLOPS	**Fl**oating-point **O**perations **P**er **S**econd
HTTP	**H**yper **T**ext **T**ransfer **P**rotocol
HPL	**H**igh-**P**erformance **L**inpack
HPL-AI	**H**igh-**P**erformance **L**inpack **A**rtificial **I**ntelligence
IC	**I**ntegrated **C**ircuit
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
IPS	**I**nstructions **P**er **S**econd
LAN	**L**ocal **A**rea **N**etwork
MPP	**M**assively **P**arallel **P**rocessor
NAT	**N**etwork **A**ddress **T**ranslator
OS	**O**perating **S**ystem
RFC	**R**equest **f**or **C**omments
TCP	**T**ransmission **C**ontrol **P**rotocol

Dedicated to all the Faculty of Applied Sciences class 2017
who made this 4-year journey.

Chapter 1

Introduction

1.1 Motivation

Smartphones are the most popular way to browse the web since 2017 (Stats, 2021a). Android OS is dominating the mobile operating system market since 2012, running on around three-quarters of all smartphones since 2017 (Stats, 2021b). The latest mobile CPU models of the most popular mobile CPU manufacturer Qualcomm (IANS, 2021) provide eight cores with up to 3.2 GHz clock speed (Qualcomm, 2021) - such frequencies are comparable with modern desktop CPUs. However, out of ten most popular use cases for smartphones, the most demanding one - gaming, resides at a sixth position (axway, 2017). Other use cases in the survey do not fully utilize all the computing power concentrated in our daily drivers.

1.2 Problem

Considering the computing performance of the most popular active distributed computing projects, scientific computing problems in fields ranging from cryptography to molecular biology require TeraFLOPS of computer performance (Wikipedia contributors, 2021). Lots of academic institutions having limited funding cannot afford computers capable of such performance. Volunteer computing allows academic institutions to reach those numbers without spending money on expensive supercomputers. Volunteer computing is a concept of donating computing resources to projects, which use those resources to do distributed computing (BOINC, 2018). Grids of Android smartphones may be responsible for a significant amount of resources in volunteer computing projects.

Smartphone users are getting more and more concerned with the time they spend using their gadgets (Trends, 2021). Apps like Forest, which gamify the process of staying out of the smartphone, have dozens of millions of installs and win multiple awards (Google, 2021). Donating smartphone computing power to a computing grid may be gamified to spend less time on a smartphone while carrying out scientific computations.

1.3 Goals

The goal of the thesis is to develop an Android middleware for grid computing. The thesis covers client application joining/leaving the computing grid, receiving and computing the work units, sending results to the server. The server responsibilities covered in the thesis are keeping track of the worker nodes, sending the work units, and collecting the results. Dividing the problem into work units and aggregating the

collected results into a problem solution are not covered in the thesis as those are not the responsibilities of the middleware.

Chapter 2

Background Information

2.1 High Performance Computing

2.1.1 Performance Measuring

Computer performance can be measured in instructions per second or floating-point operations per second. In computational science, FLOPS is a more accurate representation of CPU numerical performance. The IEEE Standard for Floating-Point Arithmetic defines five basic floating-point number formats, two of which, binary 32-bit (single precision) and binary 64-bit (double precision), are widely used ("IEEE Standard for Binary Floating-Point Arithmetic" 1985). For this thesis, FLOPS is the number of 64-bit floating-point operations per second, as it is in the TOP500 supercomputer list.

2.1.2 TOP500

The TOP500 table biannually shows the 500 most powerful supercomputers in the world. The performance of the supercomputers is measured using the LINPACK benchmark. Since June 2005, every supercomputer on the list has achieved at least one teraflops of performance. Since June 2019, every supercomputer on the list has achieved at least one petaflops of performance (TOP500, 2020c). Although the fastest supercomputer (as of November 2020) on the list, Supercomputer Fugaku, achieved 442,010 teraflops in the LINPACK benchmark (TOP500, 2020b), it reached 2 exaflops in the HPL-AI benchmark (Dongarra, 2020).

2.1.3 Parallel Computing

Moore's law predicts that the number of transistors in an integrated circuit doubles about every two years (Moore, 1975). Along with the number of transistors in an IC, computer performance should also grow exponentially. Since 1993, the performance of the №1 TOP500 supercomputer has roughly doubled every 14 months (TOP500, 2020c).

CPU performance increases from the mid-1980s until 2004 were achieved mainly by scaling its frequency. After the end of the frequency scaling, Moore's law was still in effect as manufacturers started to increase the number of CPU cores. As a result, parallel computing has become the dominant paradigm in computer architecture in the form of multi-core processors (Asanovic et al., 2006).

A massively parallel processor is a computer with multiple processors connected via the network. MPP architecture is the second most popular TOP500 supercomputer architecture (TOP500, 2020a).

2.1.4 Distributed Computing

Contrary to parallel computing, in distributed computing, the processors do not have access to a shared memory; processors communicate by passing messages. As a result of the COVID-19 pandemic, one of the biggest distributed computing projects, Folding@home, surpassed one exaflops threshold making it the world's first exascale computing system (Hruska, 2020).

A computer cluster is a set of network-connected computers that behave as a single distributed system. The most common way to set up a cluster is to connect homogeneous computers using LAN. Cluster architecture dominates system and performance share in the TOP500 supercomputer list (TOP500, 2020a).

A computing grid is geographically dispersed, usually heterogeneous computers connected by a global network that form a distributed computing system. Contrary to a computer cluster, computer grid nodes are loosely coupled. The name "grid" comes from the definition, which states that the computing resources should be available as electricity in power grids. Because of limited connectivity between computing nodes, grid computing deals mainly with embarrassingly parallel problems, which do not require internode communication. Because of nodes' high volatility, the opportunistic approach is the most popular, where work units are matched randomly, and there are no guarantees regarding the availability of resources at a given time. In a quasi-opportunistic approach, nodes are coordinated to achieve an increased quality of service. Grids are often constructed with general-purpose middleware.

Jungle computing is a form of high-performance computing where computations are offloaded across a cluster, grid, and cloud computing (Seinstra et al., 2011).

2.2 Android Operating System

2.2.1 Processes and Application Lifecycle

Unless specified otherwise, the Android application runs its own Linux process. The process is created when some application code needs to be run and will remain running until the user no longer requires it or the system needs to reclaim its resources.

A fundamental feature of Android is that the application does not directly control its process's lifetime. The application process's lifetime is determined by the Android OS considering the parts of the application that are running, how important those are to the user, and how many system resources are available at the time.

A process importance hierarchy exists to determine the order in which processes will be killed in case of low system resources. Application process importance hierarchy divides every process into one of the four categories (in order of importance):

1. A foreground process is the one with which the user is interacting, which is at the top of the screen.

2. A visible process is doing the work that the user is aware of. For example, if there is a dialogue window before some application, then the dialogue window is part of a foreground process, and that application is running in a visible process.

3. A service process is not directly visible to the user. However, it is doing things that the user cares about - background network data upload/download, for example. Services running for a long time (such as 30 minutes or more) may

be demoted in the importance hierarchy to allow their process to drop to the cached list. This helps avoid situations where long-running services that use excessive resources prevent the system from delivering a good user experience.

4. A cached process is one that the system considers not currently needed for the user, so the system is free to kill it when resources are needed elsewhere. Usually, these are the only processes involved in resource management: a well-running system will have multiple cached processes available for efficient switching between applications and regularly kill the oldest ones as needed.

(Developers, 2020)

Chapter 3

Related Works

3.1 Folding@home: Lessons From Eight Years of Volunteer Distributed Computing

Folding@home (Beberg et al., 2009) is the volunteer computing project that was the first to achieve the exaflops scale performance. Released in the year 2000, it faced lots of problems with its initial design. Being developed before the establishment of NATs and firewalls, Folding@home volunteers had issues with connecting to the work server. Work servers are stated to be a system bottleneck so that in 2004 collection servers were introduced to collect finished work units in case of work server failure. Folding@home distributes work unit cores as a platform-specific binary. Optimizing by hand assembly code for each CPU indeed results in significant speedups. However, the need to optimize the code manually slows down the introduction of the new platforms. Moreover, it may result in a loss of the noticeable amount of volunteer power after a major system update on volunteer machines. Application checkpointing is a feature that minimizes the amount of work lost in case the client goes down. This feature is not used to its full extent. The project can be improved by sending the already computed work unit state on the client's graceful shutdown to the backend, which will assign another client to continue computations.

3.2 BOINC: A System for Public-Resource Computing and Storage

Released in 2002, BOINC (Anderson, 2004) is open-source middleware operating public-resource computing projects. Comparing to Folding@home, BOINC has a more mature architecture. It makes it easy to create a work unit and support multiple applications. The client application can be run in multiple ways to tailor for specific requirements of the volunteer. BOINC introduces support for a redundant computing mechanism for identifying and rejecting erroneous results. Furthermore, it provides a homogeneous redundancy feature to force executing the work units only on the same platforms to avoid differences in results caused by differences in platforms.

Focus on rewarding the participants forces BOINC to make unnecessary operations. When redundant computing is enabled, once the quorum establishes the canonical result, clients still computing will continue to do so, and their results will be validated to reward participants.

3.3 Middleware for Grid Computing on Mobile Phones

The work (Masinde, Bagula, and Ndegwa, 2010) designs a grid computing middleware for mobile phones. The role of the server control plane belongs to one of the phones in the grid. If this phone leaves the grid, the calculations are lost because the system state is not replicated. The system relies heavily on peer-to-peer communications, which is hard to achieve with geographically disperse nodes that communicate over the global network because of the NATs prevalence.

Chapter 4

Proposed Solution

Throughout this chapter, the terms client/worker/node and server/control plane are used interchangeably, having the same meaning.

4.1 Android Application Considerations

Unlike desktop computers with a constant power supply, Android smartphones rely on a battery. Android smartphones also do not always use an unmetered network connection like Wi-Fi. In these cases, the mobile data plan charges may apply. To avoid draining volunteer's battery and paying for metered networks popular Android solutions like BOINC and DreamLab work only when the device is connected to Wi-Fi and power supply (BOINC, 2020; Vodafone, 2021). In such a case, smartphone availability for the cluster is limited not only by these conditions but also by the times when the user consciously opens the app. The latter condition is proposed to be the single one. If the user consciously launches the app, the smartphone should be available for the grid even on a low battery and connected to a metered network. Considering the specifics of the Android process lifecycle (see 2.2.1), it is recommended to leave the app running in the foreground to avoid it being killed.

4.2 Network Topology

Worker nodes are connected to the control plane in a star network topology: there is only a connection between the nodes and the control plane. The system does not provide direct internode communication routes. Moreover, it may not even be possible because of the NATs.

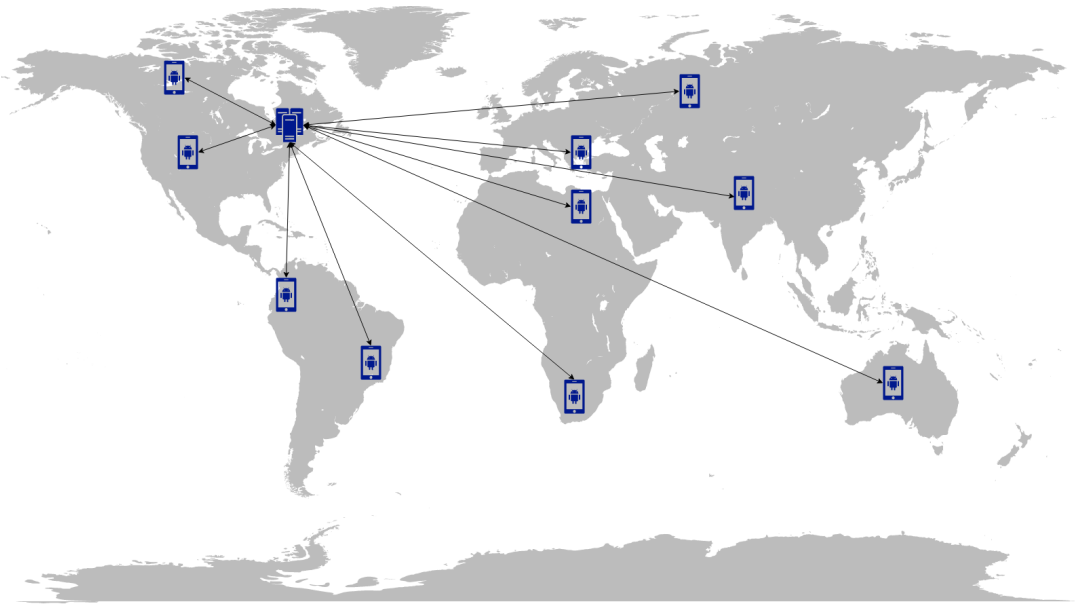

FIGURE 4.1: Star network topology: every worker node connects only to the control plane.

4.3 Redundant Computing

Due to the high volatility of Android smartphones, the smartphone may become unavailable during the work unit computations. Thus redundant computing is required. The work unit is sent to a randomly selected $2n + 1$ idle worker node to compute. As soon as the control plane receives the $n + 1$ similar result, it is accepted as a canonical result. If the timeout passes or every worker submits its result and there is no quorum, the worker unit is resent to a new set of random $2n + 1$ idle worker nodes (the sets may interfere, but it is doubtful in case there is a massive number of nodes). The results obtained in a previous work unit worker group are not considered while selecting a canonical result from a new work unit worker group. If the work unit cannot get a canonical result after k cycles of reassigning the work unit to a new work unit worker set, the control plane considers the work unit invalid and drops it from a queue.

4.4 Bidirectional Client-Server Communication

4.4.1 Push Model

Existing solutions rely on clients pulling (requesting) the work unit from the server (Beberg et al., 2009; Anderson, 2004). Unless the client automatically requests the new work unit after completing the previous one, the system loses computational power. The push model does not have such drawbacks. Moreover, by forcing clients to change the work unit they are working on, it is possible to avoid computing units that have already established the canonical answer. To enable the push model the bidirectional client-server communication is required. To provide bidirectional communication, WebSocket protocol is proposed.

WebSocket protocol provides a full-duplex communication channel over a single TCP connection. Although WebSocket and HTTP are different, WebSocket is designed to work over HTTP ports and support HTTP proxies and intermediaries, thus making it compatible with the HTTP protocol (Fette and Melnikov, 2011). The WebSocket handshake uses the HTTP Upgrade header to change from the HTTP protocol to the WebSocket protocol to achieve compatibility.

4.4.2 Healthchecks

Healthchecks are implemented using WebSocket control frames. WebSocket control frames have a payload length of 125 bytes or less (Fette and Melnikov, 2011), so they use only a tiny part of the available bandwidth. With the interval in t_0 milliseconds server sends Ping frames to the client. If the client responds with the Pong frame, the server considers it alive. If there is no Pong response during k subsequent Ping requests (kt_0 milliseconds), the client is considered dead. After m, where $m > k$, subsequent Ping requests (mt_0 milliseconds) to the dead client, server closes the connection.

4.4.3 Client-Server Communication

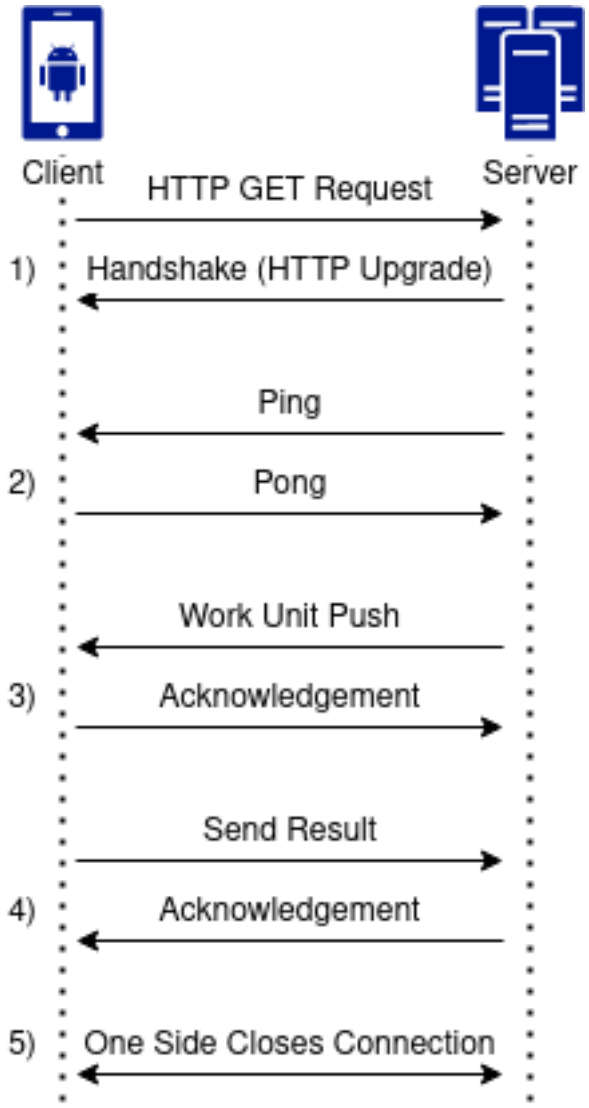

FIGURE 4.2: Client-server communication.

1. The client joins the grid by sending the HTTP GET request. The server answers upgrading the HTTP to the WebSocket protocol.
2. After initializing under the grid, the server continuously sends Ping frames to determine the client's state.
3. The server sends the work unit for the client to compute. Client answers confirming the receival of the work unit.
4. The client sends the computed result of the work unit to the server. Server answers confirming the receival of the result.
5. The client closes the WebSocket connection if it decides to leave the grid voluntarily, or the server closes the connection if the client is considered dead for a long time.

4.4.4 Worker Node Lifecycle

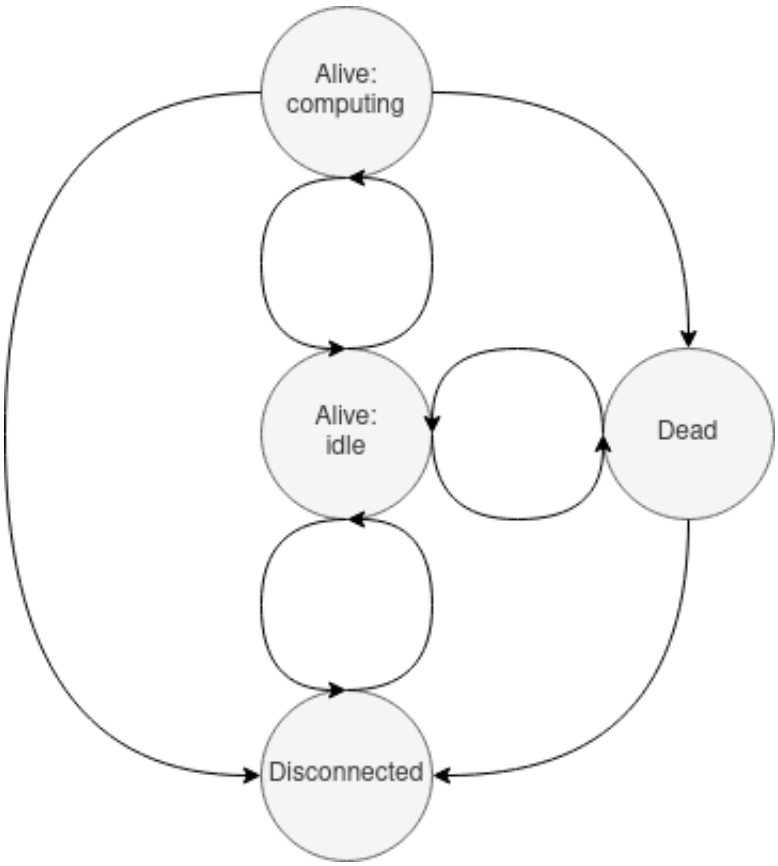

FIGURE 4.3: Worker Node Lifecycle.

TABLE 4.1: Worker node states.

State	Condition
Alive	The client answers the Ping frames from the server.
Computing	The client is alive computing the work unit, and its work unit does not yet have a canonical result.
Idle	The client is alive and its work unit has accepted a canonical result, or the client does not yet have a work unit to compute.
Dead	The client is not answering the Ping frames.
Disconnected	The alive client may voluntarily leave the computing grid, or the server closes the connection if the client is dead for too long.

4.5 Security

Work units and results are encrypted in transfer by WebSocket Secure. Checksums of the work units and results are provided to verify the data integrity.

No single node in a grid can be trusted. Thus redundant computing (see 4.3) is applied. However, relying on the majority makes the system vulnerable to the 51-percent attack vector.

The control plane adds excessive random data to the work unit before sending it to the worker. Thus checksums of the same work unit will differ from worker to worker, making it harder for the malicious workers to check whether they are in the same work unit group.

4.6 Fault Tolerance

The fault tolerance of the control panel is provided by replicating the server instances load balancing between them. Database that holds the system state is distributed across multiple servers.

Worker nodes are replicas of each other, so the worker nodes are fault-tolerant if enough are alive to take a work unit.

Chapter 5

Conclusion

This work examines the models of distributed computing and available on the market systems for volunteer computing. The shifting from the pull model to the push model in grid computing is proposed. A grid computing system for Android smartphones is designed with their specifics in mind.

The work can be extended to achieve a quasi-opportunistic system to make it suitable for problems that cannot tolerate a computing node failure. Decentralizing the system is also possible by removing the control plane, moving its responsibilities to the worker nodes, and enabling peer-to-peer communication.

Bibliography

Anderson, David P (2004). "Boinc: A system for public-resource computing and storage". In: *Fifth IEEE/ACM international workshop on grid computing*. IEEE, pp. 4–10.

Asanovic, Krste et al. (2006). "The landscape of parallel computing research: A view from Berkeley". In.

axway (2017). *10 Years On From iPhone® Launch, Axway Survey Examines Consumer View of Smartphones*. URL: https://www.axway.com/en/company/media/2017/press-release-10-years-iphoner-launch-axway-survey-examines-consumer-view (visited on 05/15/2021).

Beberg, Adam L et al. (2009). "Folding@ home: Lessons from eight years of volunteer distributed computing". In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, pp. 1–8.

BOINC (2018). *Volunteer Computing*. URL: https://boinc.berkeley.edu/trac/wiki/VolunteerComputing (visited on 05/15/2021).

— (2020). *Android FAQ*. URL: https://boinc.berkeley.edu/wiki/Android_FAQ (visited on 05/15/2021).

Developers, Android (2020). *Processes and Application Lifecycle. Android Developers*. URL: https://developer.android.com/guide/components/activities/process-lifecycle (visited on 05/15/2021).

Dongarra, Jack (2020). *HPL-AI. Results*. URL: https://icl.bitbucket.io/hpl-ai/results/ (visited on 05/15/2021).

Fette, Ian and Alexey Melnikov (2011). *The websocket protocol*.

Google (2021). *Forest: Stay focused - Apps on Google Play*. URL: https://play.google.com/store/apps/details?id=cc.forestapp (visited on 05/15/2021).

Hruska, Joel (2020). *Folding@Home Crushes Exascale Barrier, Now Faster Than Dozens of Supercomputers. ExtremeTech*. URL: https://www.extremetech.com/computing/308332-foldinghome-crushes-exascale-barrier-now-faster-than-dozens-of-supercomputers (visited on 05/15/2021).

IANS (2021). *Qualcomm leads smartphone application processor market, Apple second | Business Standard News*. URL: https://www.business-standard.com/article/international/qualcomm-leads-smartphone-application-processor-market-apple-second-121032100139_1.html (visited on 05/15/2021).

"IEEE Standard for Binary Floating-Point Arithmetic" (1985). In: *ANSI/IEEE Std 754-1985*, pp. 1–20. DOI: 10.1109/IEEESTD.1985.82928.

Masinde, Muthoni, Antoine Bagula, and Victor Ndegwa (2010). "Middleware for Grid Computing on Mobile Phones". In: *M-Science: Sensing, Computing and Dissemination*. International Centre for Theoretical Physics, Trieste, Italy.

Moore, Gordon (1975). *Progress In Digital Integrated Electronics*. URL: http://www.eng.auburn.edu/~agrawvd/COURSE/E7770_Spr07/READ/Gordon_Moore_1975_Speech.pdf (visited on 05/15/2021).

Qualcomm (2021). *Snapdragon 8 Series Mobile Platforms*. URL: https://www.qualcomm.com/products/snapdragon-8-series-mobile-platforms (visited on 05/15/2021).

Seinstra, Frank J et al. (2011). "Jungle computing: Distributed supercomputing beyond clusters, grids, and clouds". In: *Grids, Clouds and Virtualization*. Springer, pp. 167–197.

Stats, StatCounter Global (2021a). *Desktop vs Mobile vs Tablet vs Console Market Share Worldwide. 2009 - 2021*. URL: https://gs.statcounter.com/platform-market-share#yearly-2009-2021 (visited on 05/15/2021).

— (2021b). *Mobile Operating System Market Share Worldwide. 2009 - 2021*. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide/#yearly-2009-2021 (visited on 05/15/2021).

TOP500 (2020a). *Development Over Time. Architecture*. URL: https://www.top500.org/statistics/overtime/ (visited on 05/15/2021).

— (2020b). *November 2020. TOP500*. URL: https://www.top500.org/lists/top500/2020/11/ (visited on 05/15/2021).

— (2020c). *Performance Development*. URL: https://www.top500.org/statistics/perfdevel/ (visited on 05/15/2021).

Trends, Google (2021). *digital detox, digital minimalism*. URL: https://trends.google.com/trends/explore?date=all&q=digital%20detox,digital%20minimalism (visited on 05/15/2021).

Vodafone (2021). *Help with COVID-19 and cancer research. FAQs for Android users*. URL: https://www.vodafone.co.uk/mobile/dreamlab (visited on 05/15/2021).

Wikipedia contributors (2021). *List of distributed computing projects — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=List_of_distributed_computing_projects&oldid=1020340090 (visited on 05/15/2021).