



ByteNite
Distributed Computing

White Paper

Release date: *January 16th, 2023*

Version: *Full, v2.1*

Table of Contents

1. State Of The Art	1
1.1. Grid vs. Cloud Computing	1
1.2. Grid Computing Today	3
1.3. Fact	5
2. What is ByteNite	7
2.1. Innovation	8
2.2. Business Model	9
2.3. Glossary	10
3. Core System	12
3.1. Architecture	12
3.2. Workflows	14
3.3. Business Logic	21
Bibliography	37
Mathematical Appendix	40

Bookmarks

[Figure 1](#) – CPU Mark, 2004 to 2022

[Figure 2](#) – Architecture diagram

[Figure 3](#) – Job submission workflow

[Figure 4](#) – Task processing workflow

[Figure 5](#) – Election of eligible devices

[Figure 6](#) – Capacity pools

[Figure 7](#) – Pool queues (1)

[Figure 8](#) – Pool queues (2)

[Figure 9](#) – Fault response workflow

[Table 1](#) – ByteNite’s glossary

[Table 2](#) – Grid state’s parameters (1)

[Table 3](#) – Grid state’s parameters (2)

[Table 4](#) – Job specification (1)

[Table 5](#) – Job specification (2)

[Table 6](#) – Job preferences

[Table 7](#) – Capacity score’s components

[Table 8](#) – Fault rate’s inputs

[Table 9](#) – Threshold flags

[Equation 1](#) – Capacity score’s formula

[Equation 2](#) – Fault rate’s formula

[Equation 3](#) – Repechage lottery’s formula

[Algorithm 1](#) – Creation of the ByteRank

[Algorithm 2](#) – Creation of capacity pools

About the author

Fabio Caironi graduated from the University of Milan in 2019 with a degree in mathematics and a discussion on *The Radon-Nikodym theorem: applications on the existence of conditional probability*. He attended the two-year Master's Degree course in Data Science and Economics at University of Milan, where he could refine his knowledge in topics of economic theory applied to data, machine learning, data management technologies, cloud environments, decision theory under uncertainty conditions and financial time series. His research focus covers machine learning techniques, distributed computing systems, healthcare and financial data analysis, and video technologies. In 2020 he co-authored a software publication on the real-time tracking and modeling of the covid-19 outbreak in Italy, dubbed disCOVIDer19. In 2021 he founded ByteNite, a commercial grid computing project exposing high-throughput applications like video encoding, machine learning, and graphics rendering. In 2021 he filed a patent application for an efficient task-scheduling method in a distributed computing system. As the company's CEO, he currently leads the project and the team, and has moved the product from idea stage to a market-ready beta.



Acknowledgments

I would like to thank all the people that made this project possible from its genesis in May 2021 to date. I am grateful for their passion and commitment in their daily work as well as in their unsolicited contributions.

Thanks to Ksenia Security S.p.A., for their investment and their friendly partnership throughout all 2022, and the interesting gatherings and exquisite Italian dinners we had all together.

Thanks to Raffaele Di Crosta and Giorgio Finaurini for their solid endorsement in this project and their continued advice and support as board members.

Thanks to ByteNite's head of development Niccolò Castelli, for turning this paper into an operating grid computing system with care and patience, and thanks to all the staff that worked on the product. They've all been essential gears of our system!

FABIO CAIRONI
CEO & Founder, ByteNite Inc.

Summary

This White Paper describes in full detail a model for a new commercial grid computing implementation called "ByteNite". I open the Paper with the state of the art of the distributed computing models, including an overview of cloud and grid computing, their commonalities and history, and how they are topical in today's world (§1. State Of The Art). I build the foundations of our work through a critical insight that triggers powerful implications in connection with the current technologies: the availability of a gigantic computing capacity inside worldwide consumers' and businesses' devices, enhanced by the cloud computing model (§1.3. Fact). I address the new proposed model through a description of the system, its overall operation, the underlying business concepts, and the innovative value proposition (§2. What Is ByteNite). I then dive into its architecture and workflow design, delineating its structure, key features, and the chronological phases of its activity (§3. Core System). The paper then reveals the main algorithm running in the core system and its inputs. This last section (§3.3. Business Logic) includes all the relevant details about how does ByteNite manage the workload, including the type of information collected from the workers and the users, the creation of specific grid indexes, a scheduling algorithm, a distribution process, and a fault tolerance mechanism.

Intellectual Property Notice

ByteNite Inc., a US corporation with offices at 708 Long Bridge Street, San Francisco (CA), is the sole owner of the white paper and all its contents. The white paper and its contents, including but not limited to the business idea, technological context, workflows, architecture, algorithms, economic and business model, and any inventions described in the paper, are the property of ByteNite Inc. and are protected by copyright laws.

Unauthorized reproduction or usage of any part of the white paper or its contents is strictly prohibited. Any reproduction or usage of the white paper or its contents must be done with the express written permission of ByteNite Inc. and must include an explicit reference to the company as the source.

Any unauthorized reproduction or usage of the white paper or its contents may infringe upon the intellectual property rights of ByteNite Inc. and may be subject to legal action. The white paper and its contents are also the subject of pending patents in the US and Europe, and any unauthorized reproduction or usage may also infringe upon these patent rights.

By accessing the white paper, you acknowledge and agree to the terms of this disclaimer and to the intellectual property rights of ByteNite Inc. Please respect the hard work and intellectual property of our company and do not reproduce or use the white paper or its contents without permission.

1. State Of The Art

In the IoT and Big Data era, cloud computing and distributed file systems are fundamental for data management and processing. Big tech firms and their server farms are the most valuable resource we can rely on today for outsourcing computations; edge computing has become indispensable in many applications as the volume of data produced daily by businesses is increasingly significant.

Years of technological advancement have paved the way to bring cloud computing towards Industry 4.0, making it possible for a wide range of cloud solutions to become a reality, bringing innovation and efficiency to business processes and changing our lifestyles. Many new businesses that operate in the cloud sector, such as Snowflake, Cloudflare, Databricks, emerged in the past fifteen years, and well-known tech industry leaders Google, Microsoft, Amazon, and IBM, could become or remain IT giants as a result of their readiness to seize the cloud's opportunity.

Cloud computing is more than renting someone else's machines: it encompasses workload management, service orchestration, distributed storage, and much more. However, it all boils down to the target machine's computing power provided by its processor when it comes to throughput and performance. After all, as B. Sosinsky [1] has said, "cloud computing is revolutionary, even if the technology it is built on is evolutionary."

With the benefit of hindsight in a fully digitalized era, have we ever tried to unwrap cloud computing and question if there is more we can learn from its foundations? Furthermore, as the on-premise commercial model has shifted to cloud computing with the advent of the internet, what will the increase in worldwide connectivity and the rise of 5G turn the cloud model into?

1.1. Grid vs. Cloud Computing

The invention described in this White Paper mostly conforms to the techniques dictated by the "grid computing" model. However, several other topics and frameworks can be deemed relevant to this invention, including utility or on-demand computing, high-throughput computing, distributed computing, and, most of all, cloud computing. Grid and cloud computing share several key traits, such as their reliance on distributed resources. Still, they differ slightly in many domains, including business model, architecture, resource management, and application model. Today, grid computing has evolved to become the basis of the more advanced cloud, offering more robust performance in a secure virtual environment. Yet, I am convinced that there is much value left behind in this transition, and no project or initiative has been able to seize and implement it at scale so far. I shall begin this introduction with a brief overview of the two paradigms, as per current scientific literature, and then review both past and current grid computing projects. I will then establish the grounds of our theory, shedding light on the immense opportunity that grid computing represents in today's technologically evolving world (§1.2), and finally lay down our value proposition (§2.1).

The term “grid computing” refers to a form of distributed computing featuring heterogeneous and geographically dispersed resources provided by different organizations. Grids were developed in the mid-1990s to provide a solution for large-scale computational tasks that required significant processing power, only affordable by supercomputers back then. According to Bote-Lorenzo et al. (2004):

“ A grid can be defined as a large-scale geographically distributed hardware and software infrastructure composed of heterogeneous networked resources owned and shared by multiple administrative organizations which are coordinated to provide transparent, dependable, pervasive and consistent computing support to a wide range of applications. These applications can perform either distributed computing, high throughput computing, on-demand computing, data-intensive computing, collaborative computing or multimedia computing. [2]

A year later, in 2005, IBM’s Introduction to Grid Computing put the grid computing definition closer to a ‘virtualization’ concept that would become the key principle of the cloud:

“ If we focus our attention on distributed computing solutions, then we could consider one definition of grid computing to be distributed computing across virtualized resources. The goal is to create the illusion of a simple yet large and powerful virtual computer out of a collection of connected (and possibly heterogeneous) systems sharing various combinations of resources. [3]

Virtualization turned out to be a big win in the utility computing model: it allowed applications to be abstracted from the underlying fabric and deployed on-demand to more exacting customers. That’s how we arrived at cloud computing. Subsequently, its rapid adoption from the mid-2000s was fostered by the decrease in hardware cost and increase in computing power and storage capacity, as well as the exponentially growing size of data and processing power used by modern internet applications and services.

Cloud computing delivers different levels of scalable and dynamically configurable services to customers outside the cloud. A comprehensive definition of cloud computing is given by one of its ancestors, Ian Foster, in his article “Cloud Computing and Grid Computing 360-Degree Compared” (2009):

“ [cloud computing is] a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet. [4]

The virtualization feature of cloud computing is key to providing the necessary abstraction to deliver on-demand computing power, storage, and networking and to meet stringent service-level agreements (SLAs) with customers. It is more difficult to find this level of virtualization in standard grid implementations, as each organization within a grid usually maintains complete control over its resources.

On the architecture level, grids and clouds share a fabric layer consisting of the raw hardware resources and the protocols to access them. While clouds provide a unified

resource layer to virtualize such resources and expose them to end-user applications, grids feature a more complex set of standard protocols, middleware and toolkits to connect and manage resources.

Finally, ensuring interoperability and security are fundamental both for grid and cloud infrastructures. While in grids interoperability comes built-in (they are based on the assumption that resources are heterogeneous and dynamic), clouds have developed stronger security policies to comply with regulatory standards. The combination of such properties in cloud-powered grid computing systems might prove a critical vision for the future of the cloud in the 2020s.

1.2. Grid Computing Today

Nowadays, most grid computing initiatives around the world have given way to more modern and service-oriented cloud computing applications. Many grid middleware implementations and grid infrastructures built in the 2000s have either ceased operating, turned into cloud projects, or been acquired by cloud computing companies.

United Devices Inc., a commercial volunteer computing company offering high-performance computing services, was sold in 2007 to Univa, a software company that developed cloud management products, which was in turn acquired by cloud software company Altair Technologies. DataSynapse was sold in 2009 to TIBCO Software Inc., a business intelligence software company, and their grid computing middleware was turned into a BI product powered by parallel computing. A different fate awaited companies like Entropia, Inc. and Popular Power, developers of distributed computing software for CPU scavenging, which were driven out of business. And so on: the list of companies born in the new millennium trying to ride the wave of grid computing is long [5]. It is no mystery why they all failed in a matter of years: while they were able to develop large-scale computing infrastructure by accessing the spare processing capacity of thousand of volunteered CPUs, these companies didn't offer any reward to their contributors. Consequently, the resource owners had no incentive for their continued contribution, and the economic model proved not scalable nor maintainable [6]. Given those years' computing and network capabilities, the only companies that managed to survive were those that were noticed and acquired by larger corporations, which could afford substantial infrastructure investments to keep up with the incoming cloud wave.

In the volunteer computing world, grids made a name for themselves in the 2000's through scientific projects that gained much attention in the academic community. Either infrastructure-based as TeraGrid [7], middleware-based like the Globus Toolkit [8, 9], or application-based like SETI@Home [10], these projects were aimed at empowering scientific research in disparate fields (Physics, Medicine, Astronomy, Mathematics, Biology), making it possible to solve computationally intensive problems that would have been difficult or infeasible to tackle using standard computers. Some historic volunteer computing projects made their way through the 21st century and are still working in 2022. Their participation was primarily motivated by non-monetary prizes, fun, fame, or collaborative advantage. If not for the economic model, they are interesting to analyze as technically feasible grid computing projects. Hence, I shall give a quick overview of them.

The most representative is BOINC [11, 12], a platform for distributed high-throughput computing where worker nodes are desktop and laptop computers, tablets, and smartphones volunteered by their owners. A fair number of applications or "projects" are

linked to BOINC and use or have used its distributed computing infrastructure to solve large-scale scientific problems that could once be tackled only by supercomputers [11, 13]. SETI@Home was the first, and was responsible with giving BOINC the popularity it later had. SETI@Home was devoted to the Search for Extra-Terrestrial Intelligence through distributed digital signal processing of radio telescope data. A week after its launch, SETI@Home scored 200,000 participants. After four or five months, it broke one million, and later reached over two million users. In 2020 the project officially ceased operations. Other remarkable BOINC-powered projects include: Einstein@Home [14] for the search of weak astrophysical signals from spinning neutron stars; World Community Grid [15] for scientific research on topics related to health, poverty, and sustainability; and Climateprediction.net [16] for climate models simulations.

Distributed.net [17] was another volunteer computing project that attempted to solve large-scale problems, and was governed by a non-profit US corporation. As of 2019, distributed.net's throughput was estimated at roughly 1.25 petaFLOPs. Recently, distributed.net has joined forces with BOINC with the aim of finding mathematical solutions to cryptographic algorithms.

Another operating volunteer computing project is HTCondor [18, 19], an open-source distributed computing software that enables the increase of computing throughput, developed at the University of Wisconsin-Madison. HTCondor provides a job queueing mechanism, a scheduling policy, a priority scheme, and a resource monitoring and management tool, and can integrate dedicated resources (rack-mounted clusters) and non-dedicated desktop machines into one computing environment.

Finally, a distributed computing project that has lately gained a broad consensus due to new discoveries regarding SARS-CoV-2 is Folding@Home [20]. The main aim of this project is to understand protein dynamics by means of statistically distributed simulations. In 2020 the computing speed of Folding@Home peaked at 2.43 exaFLOPS, a power in the order of one billion billion floating point operations per second, or enough to mine a Bitcoin in ten seconds.

Although these projects are of great help for research, they won't be able to unlock the full potential of a worldwide grid. Their genesis and purpose keep them away from reaching a wider audience and becoming marketable products. The replicability of any of these models on the market is not only prevented by the lack of a well thought-out payment framework, but especially by the lack of a performance-oriented resource management system built with modern and widely adopted standards and protocols.

Starting in 2010, a new distributed technology started bringing collaborative computing back into the spotlight. A new global paradigm was established and many companies followed by building products on top of it, or creating private sub-networks to capitalize on what proved to be more than a brand-new concept. I am referring to the blockchain and all the blockchain-powered dApps (decentralized applications) that have been implemented thanks to the wild proliferation of this technology. A dApp is an open-source software application that runs on a peer-to-peer blockchain network. dApps are built for disparate use cases across various industries, including finance and payments, gaming, supply chain, user-generated content networks, and distributed computing.

The latter use case is relevant to our framework, as it involves dApps that exploit member devices' processing power and network to improve and democratize access to CPU- or GPU-intensive digital services. Some notable implementations of decentralized computing involve video streaming (Livepeer [21, 22], Theta Network [23]), mobile blockchain mining (Sweatcoin [24], MinePi [25]), and general-purpose computing (Golem [26], Cudos [27, 28], iExec [29]).

These applications usually use Ethereum or purpose-minted coins for collecting and distributing payments, and they handle crypto transactions and task validation with smart contracts. Ethereum also provides these dApps solutions for guaranteeing distributed consensus and identity management.

A question that might arise is how Ethereum and, generally, blockchain technology actually empower distributed computing on the processing side. The answer is simple: it doesn't. Uriarte, R.B. and DeNicola, R. (2018) [30], from IMT School for Advanced Studies of Lucca, have analyzed the architectures of three blockchain-based decentralized cloud solutions. Their finding is that in all three projects, smart contracts, payments, and reputation are managed in a "transaction network" built on the blockchain, while the actual computing services are executed in a "side-chain network" charged with processing, negotiation, and verification of computing tasks. As the paper highlights, the results obtained from a collaborative, distributed computing network might be chaotic and heterogenous; hence, the side-chain network reveals a non-deterministic behavior that must be mediated in order to reach a consensus in the transaction network, and a specific component is needed to interface between the two networks. This adds complexity to the already high computational cost of running and maintaining a blockchain.

There are other elements holding back Ethereum and other blockchain technologies from implementing a large-scale, efficient grid like the one discussed in this White Paper. Two of them are the high transaction costs and the capped transaction throughput (Ethereum can process less than 30 transactions per second), which both pose serious threats to performance and scalability. Another shortcoming is the almost absent definition of Quality of Service in most dApps' smart contracts, or even in their general terms and conditions. Besides signaling an inability to control and measure the average processing performance, the absence of QoS makes big customers, which are seldom unconcerned with quality guarantees, shy away from blockchain-powered computing solutions.

Finally, it is worth mentioning that, despite being the core philosophy of such dApps, the restriction to support only crypto wallets and cryptocurrency transactions cuts off the vast majority of both resource providers and cloud computing customers, who normally do business with fiat currencies and are still – and possibly forever – crypto-averse.

1.3. Fact

In 2023, an immense underlying computational power is widespread throughout the globe and sits idle most of the time. Altogether, it overcomes the joint processor capacity of the biggest cloud providers by tens of times.

More than 12 billion computers, smartphones, tablets, and other commercial electronic devices are hiding immense potential, especially now that they're shipped with ever more performing hardware (see [Figure 1](#)), and are unexploited during the inactivity of their human owners, like during the night. Not only are electronic consumer devices underused: many businesses owning disparate hardware, from video production facilities to private data centers and office desktop computers, don't know how to use it when it's not at work.

Past and existing grid computing projects have shown us the potential of building a distributed computing farm by tapping into a category of machines not originally sold to fulfill utility computing purposes – the mass consumer technology. However, such a vast unused computational power couldn't be easily gathered and connected until a few years

ago because of major technological limitations, including the average network speed, network coverage, and the hardware capacity of common devices on the market. In addition, the attempts to build a global grid have been held back by exclusively technology-gearred visions and major market misunderstandings, largely attributable to shortsighted or too-technical founders, that entailed failing execution strategies and limited outcomes.

Today, the easy and fast access of any device to the internet and the virtualization provided by the cloud make it possible to collect and utilize the vast worldwide computing potential in a distributed computing system, reviving the already-known paradigm of grid computing and enhancing it with the reliability, scalability, and automation provided by the cloud. However, the lessons learned from the past make us steer clear of development strategies that have grid technology as the only guiding star. For such a massive commercial project to be successful, any development choice, from architecture to applications, must be driven by evident market demands and clear economic visions, that spur the adoption of grid computing as key to solving market-inherent cost-benefit problems.

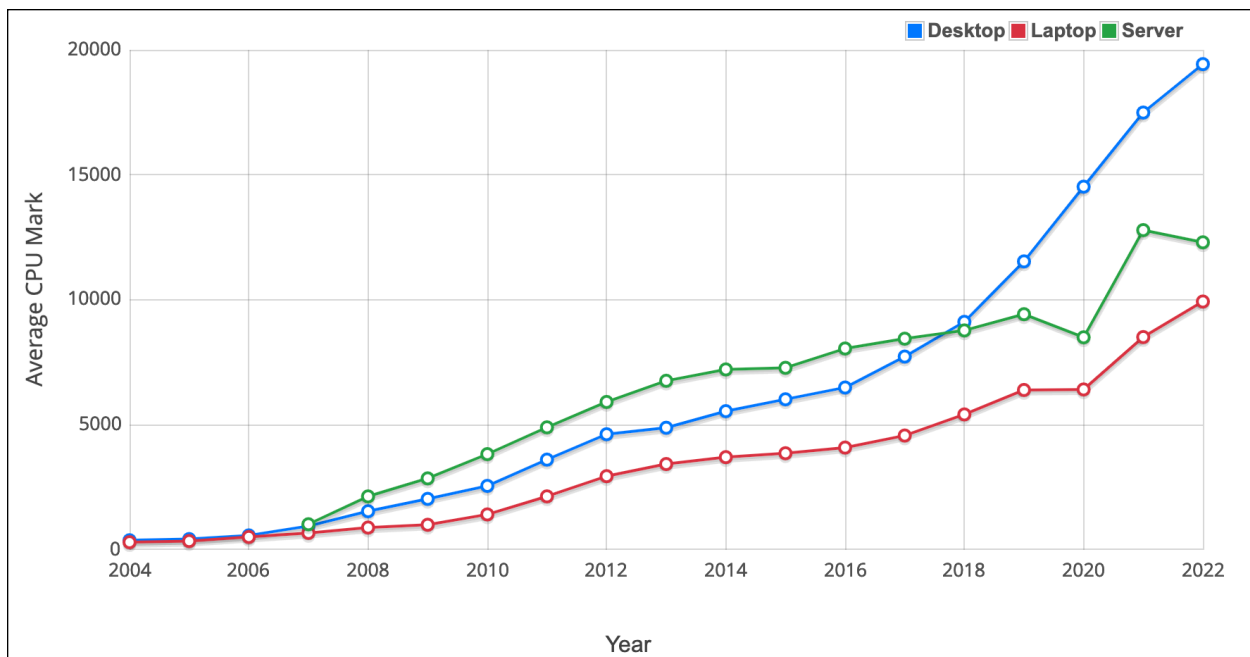


Figure 1 – CPU performance of desktop, laptop and server computers from 2004 to 2022. Courtesy of [PassMark Software](#).

2. What is ByteNite

ByteNite is a commercial, centralized, service-oriented grid computing system based on subscriber devices' processing capacity, realizing a high-throughput computing environment for utility computing purposes. Rather than an online marketplace, where buyers and sellers are directly put into contact, ByteNite creates two different and separate hubs that are accessible by the purchasers of computing services ("users" or "customers") and by the suppliers of computing power ("workers" or "suppliers"), respectively, brokering the management of computational resources to keep the two segments well coordinated and functioning.

The three components that build up ByteNite's grid computing system are the following:

- **Core System:** the core middleware, or backend layer, responsible for managing, scheduling, retrieving, transforming, transitioning, sending, organizing, and validating the users' computational jobs. It stores and makes accessible the users' and workers' data, including job history, activity, wallet balances, and device info. It also generates quotes, collects users' payments, and distributes rewards to workers.
- **ByteNite Computing Platform:** a user-level middleware available as a software-as-a-service platform, accessible through a web UI or an API, exposing both ready-made and custom-made computing services ("applications") to customers. On the platform, users can configure, submit, and pay for computing jobs, as well as upload and download their data (inputs and outputs), and view their job history, jobs states, and summary usage. They can automate the execution of their jobs via recurring tasks and automation pipelines.
- **ByteNite Worker App:** software that runs on workers' devices and enables them to receive, queue up, process, send back, and clear up computing tasks, according to programs shipped with each task and running inside the App. The Worker App also makes available the summary of completed tasks and their credits; hence, it allows workers to redeem their credits by converting them into several forms of reward, including cash.

In other words, ByteNite provides software to connect the users to the system, schedule the workload, and connect the computational grid to the system. The workers supply the fabric layer consisting of distributed computing resources, and users provide all the inputs that feed the applications, including data.

ByteNite stands in the market as a provider of high-throughput computing services. It targets small- and medium-sized companies seeking faster performance at more affordable prices than the cloud, and enterprises that operate with big volumes of data daily who need to speed up their workflows. In both cases, ByteNite helps fulfill performance goals for specific applications that generate loosely coupled or independent tasks.

ByteNite will develop three target applications that represent its core mission and an extraordinary market opportunity: Video Encoding (market size: \$1.07B, 4.26% CAGR), Graphics Rendering (\$2.59B, 20.9% CAGR), and Computer Vision (\$12.72B, 16% CAGR). In addition to being three of the most intensive commercial computing activities, these applications are well-suited for distributed computing as each of them generates workloads that can be divided into multiple, independent smaller tasks.

ByteNite's customers will be also provided with the tools to develop their own distributed applications to run on the grid resources using ByteNite Computing Platform. We can find a variety of use cases for such tailor-made solutions in the media & entertainment industry, as well as in the financial and healthcare sectors.

On the other side, ByteNite offers a chance to make passive income out of ordinary devices, like personal and office computers, smartphones, tablets, small servers, and eventually a wider range of IoT devices like video game consoles, TVs, home appliances, and industrial electrical machinery. Whilst in 2022 we have online marketplaces to effortlessly sell or rent out almost everything from material belongings to volatile goods like electricity, it is not yet possible to rent out our devices' exceeding computing capacity in the matter of a few minutes. ByteNite brings together technology to enable such a monetization possibility with a smooth onboarding of the workers, by streamlining the workflow and condensing all the interactions into a single piece of software, the ByteNite Worker App.

2.1. Innovation

ByteNite is the first distributed computing solution to combine the following accomplishments:

- Uses heterogeneous, cross-platform, both mobile and desktop devices located anywhere as worker nodes;
- Creates a computing-capacity sharing economy based on the trade of distributed processing tasks with real money;
- Is open to everyone;
- Constantly monitors performance and automatically turns it into business requirements and price adjustments;
- Manages non-deterministic behaviors with a centralized scheduling system based on both a-priori and a-posteriori fault-tolerant techniques.

ByteNite has the mission of becoming the first worldwide grid powering a general-purpose high-throughput computing system, where everybody can build and run their own distributed applications or use ready-made flagship computing products.

ByteNite's values can be described as follow:

- **Availability**

The extension of ByteNite's grid, together with its devices' diversification, geographical distribution, and heterogeneous connectivity, allows and guarantees flexible provisioning of computing resources at any time.

- **Agility**

The commodification and customization of computing services, plus the existence of an optimal delivery pipeline, make the entire process from data ingestion to output upload extraordinarily agile.

- **Speed**

The more nodes in the grid, the less time is needed to process partitioned jobs. This fact makes ByteNite competitive and preferable to the classic cloud and on-premise computing for various use cases.

- **Sustainability**

Deploying distributed computations on existing and commonly active devices is an environmentally-friendly alternative to using server farms, provisioning new hardware, and building new infrastructure. ByteNite's distributed computing model guarantees an inherent heat dispersion from devices' processors that are connected from different locations, which eliminates the need for artificial cooling of rack-mounted servers. In addition, old or unused devices can be turned into ByteNite "workers" instead of winding up in the trash, thereby lowering the pollution caused by electronic waste.

- **Security**

Data is at the core of ByteNite's business, and so is cybersecurity. All data coming to and from ByteNite's system is encrypted and handled in isolated runtime environments, and workers are constantly monitored and readily excluded if deemed potentially malicious. In addition, ByteNite's reliance on a robust and certified cloud grants it ready and updated cybersecurity policies and implementations that are now standard for all cloud-based software companies.

2.2. Business Model

ByteNite creates an ecosystem where buyers of computing services can find solutions, and hardware owners can receive compensation for supplying their devices' computing power. To enable such a marketplace, ByteNite charges customers on a pay-per-use basis: the proceeds are then divided into a "user rewards" share (70%) and a ByteNite share (30%). Given ByteNite's cloud infrastructure costs, like egress, storage, and peripheral computing, a fair estimate of the gross profit margin can be 20%.

To keep track of inbound and outbound transactions in all currencies and fulfill internal bookkeeping, ByteNite introduces a utility currency, or fake coin, called "ByteChip", whose value can be altered at any time to keep up with economic developments. The value of a US dollar in ByteChips (€) on the date of this publication is:

$$\text{€ } 200 = \$1$$

ByteChips are used throughout ByteNite's system to purchase and sell computing services. Users can buy ByteChips on the Computing Platform and spend them on computing services. Every application has its pricing table or formula, and users can simulate the cost of their jobs through the platform.

On the other side, workers accrue ByteChips for every completed task. Every task has a "bounty" attached, which expresses the amount of ByteChips due to the worker for correctly processing it. The bounty, or prize, is proportional to the relative size of the task's input chunk to the total job's data, according to this formula:

$$bounty_i = (job\ price \cdot 0.7) \cdot \frac{size_i}{job\ size}$$


Equation 1 – Formula for the i^{th} task bounty. The term in parentheses represents the total workers' reward share for a given job; $size_i$ is the dimension of the i^{th} chunk; $job\ size$ is the dimension of the job's data.

Failed tasks don't spawn rewards, while replicated tasks bear a portion of the total task bounty. Finally, workers can use their ByteChips to purchase affiliated items on the Worker App, like gift cards, discounts, and online subscriptions, or they can redeem money in their currency via a bank transfer from ByteNite.

ByteNite uses Stripe to handle money transactions from the users to ByteNite and from ByteNite to the workers, outsourcing all the issues related to tax, currency conversion, and anti-fraud to said service.

2.3. Glossary

Before diving into the system's description, I shall provide a list of terms to minimize possible ambiguities throughout the text and help the reader get acquainted with our terminology.

<i>Term</i>	<i>Description</i>	<i>Example</i>
active devices	The devices that are currently available, or equivalently have a ByteRank greater than 0	
active grid	The portion of the grid composed of active devices	
application	same as computing application	
ByteChip 	ByteNite's utility currency, used by customers to purchase computing services, and by workers to redeem rewards	A user can buy a \$0.30-worth job using 60 ByteChips. A worker can convert 3K ByteChips into a wire transfer of \$15.
capacity pool	Group of devices selected to process a fixed amount of tasks. Changes from job to job.	A group of 45 active devices, with a capacity score spanning from 2800 to 1700
computing application	Any distributed computing program chosen or submitted by a customer, and eventually run in the grid through the Worker Apps	A Python program implementing video encoding with FFmpeg
Computing Platform	ByteNite's software-as-a-service product where users can submit their high-throughput computing jobs	app.bytenite.com
Core System	ByteNite's backend system, responsible for task management, data storage, payments, device monitoring, and much more	
chunk	A portion or segment (typically small) of the input data	10s of video

<i>Term</i>	<i>Description</i>	<i>Example</i>
customer	Like “user”, but with a connotation of physical buyer person	The chief engineer at a video streaming company, or a private visual content creator
grid	The network of worker devices intended as a virtual collection of computing nodes equipped with middleware	
job	A unit computational goal submitted by a user, expecting an input and an output	The encoding of a video file
node	A worker device or a group of tightly coupled worker devices owned by the same worker	A set of 10 office computers owned by worker “John B.”
supplier	Same as “worker”	
target subset	The list of devices selected for a job after they have passed the eligibility test	
task	A unit computational activity destined for execution on the worker devices, comprising a chunk, an executable or program, and additional metadata	A .zip file containing 10s of video, the video encoding program, and summary information about the original video
task bounty	An amount of ByteChips representing the value of a task, that is credited to a worker’s wallet upon completing the task	A 15-second video encoding task worth 2 ByteChips
user	The digital identity of a buyer of ByteNite’s computing services	An ID record like “5ZSWt76”
worker	The party supplying the computing power of its device(s) via the Worker App	A 20-year-old girl owning a smartphone and a computer, or Intel Corporation
Worker App	The software running on the worker devices that allow them to process ByteNite’s tasks and the workers to cash out the rewards	
worker device	Each of the devices that run the Worker App	A Samsung Galaxy A53 5G, or an HP Windows computer

Table 1 – ByteNite’s dictionary. The terms listed here refer to specific subjects, transactional items, or programming components that pertain ByteNite’s system and products

3. Core System

In this section, I shall give an overview of how ByteNite works from a backend perspective: how its Core System is structured, which components are responsible for running the services, what the most relevant workflows are, and how monitoring and scheduling mechanisms work.

3.1. Architecture

ByteNite's Core System has a micro-services architecture. Each service represents an independent and scalable backend component running in the cloud and interfacing with the Worker App, the Computing Platform, and the other components through dedicated APIs. The architecture diagram is depicted in [Figure 2](#).

The following internal services run the business logic (see [§3.3](#)), and are not exposed publicly:

- The *Partitioner* verifies the integrity of data uploaded by users through the Computing Platform, and splits it into smaller chunks suitable for worker devices. A task record is created for every chunk, and the record ID is queued on a job-specific Redis queue.
- The *Feeder* manages and supervises the entire task scheduling system. It takes tasks from job-specific queues and puts them in a global task queue ready to be consumed by the Tasks API. Tasks are sorted according to a scheduling algorithm ([§3.3.4](#)) that considers the availability of computing resources in the grid, the job's requirements, and the user's preferences.
- The *Validator* verifies the integrity and correctness of results sent by the worker apps. Different jobs could use different validators.
- The *Assembler* collects completed and validated tasks from the Validator and assembles them into larger chunks until it has rebuilt the full processed data file, which is uploaded to a cloud storage bucket accessible from the Computing Platform.
- The *Reward System* is responsible for clearing ByteChip transactions between ByteNite and the workers and ensuring that all balances are constantly updated.

The customer APIs handle communication with the Computing Platform:

- The *Jobs APIs* allow the Computing Platform to create and configure new jobs, send input data, send and receive state updates, and fetch download links.
- The *Billing API* allows the Computing Platform to access billing and payment information.

Similarly, the worker APIs connect the Core System with the Worker Apps:

- The *Tasks APIs* allow the Worker App to fetch new tasks, download data and programs, and send back results or abort the task.

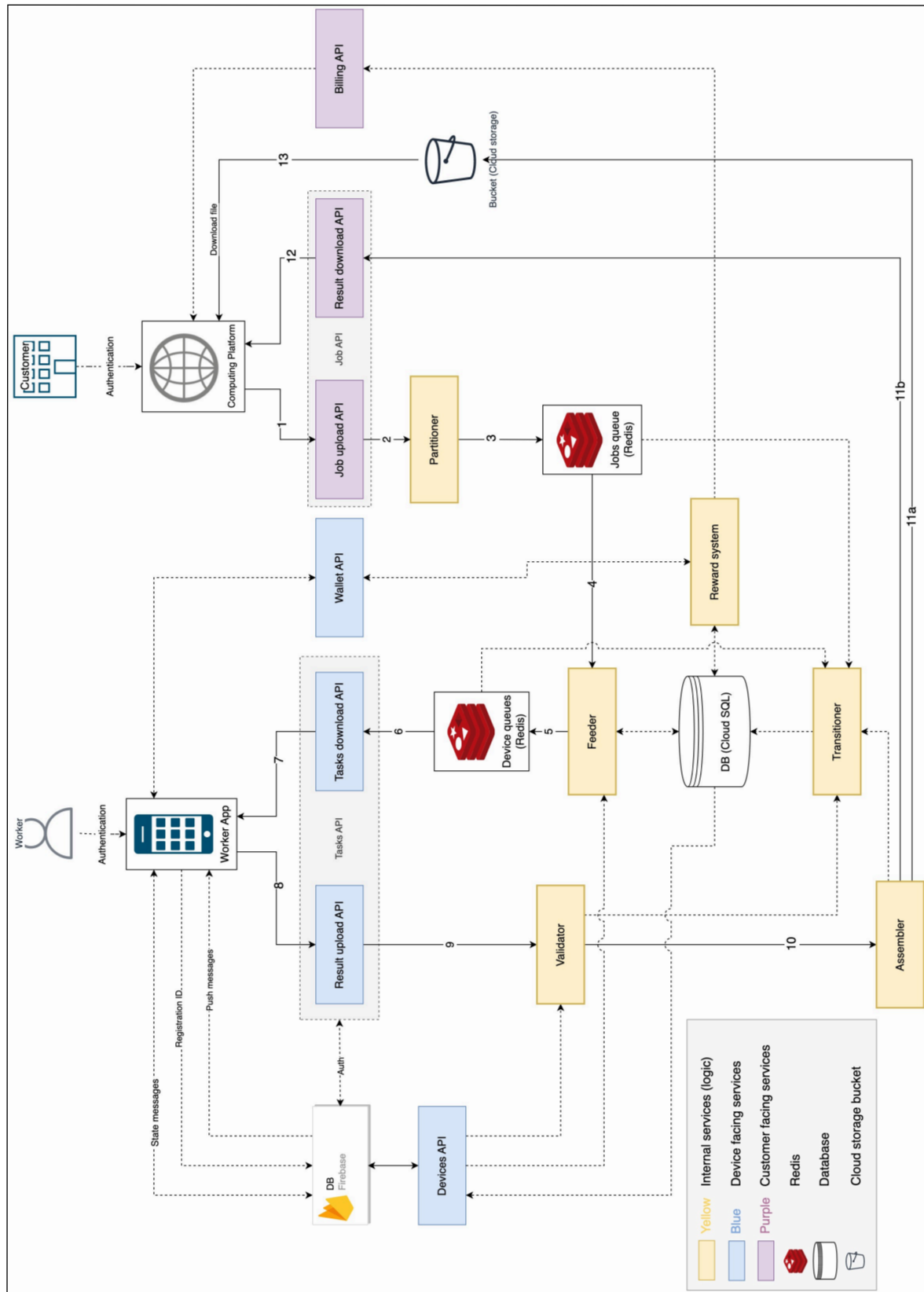


Figure 2 – ByteNite's Core System's architecture diagram

- The *Wallet API* allows the Worker App to get the ByteChip balance and history. It can also request and record ByteChip expenditures in services or payouts.
- The *Devices API* connects to Firebase to fetch information about task and device states, user authentication, and device preferences. This is the only server-side component that connects to Firebase.

Finally, ByteNite’s data is sorted and stored in the following components:

- The *Cloud SQL Database* is a SQL database that supports atomic transactions. It stores all data with persistence and consistency priorities over access performance.
- The *Firebase Database* stores all device-related information like hardware specifications and device state and handles authentication. This is the only database that directly interfaces with the devices.
- The *Redis Databases* are fast databases for internal usage that handle short-run storage for frequent reads, writes, and inter-service messages.
- *Cloud buckets* are web-based folders with access restrictions that store files downloaded or uploaded by users.

3.2. Workflows

ByteNite fulfills its twofold mandate of collecting users’ jobs and distributing them to the grid through several recurring workflows. Each workflow is a set of rules and actions happening either in the Core System, on the Computing Platform, on the Worker App, or across them. Workflows are well-coordinated with the other processes and designed to make the whole execution fault-tolerant and agile. From a 360-degree perspective, the processing of a job can be summarized as follows:

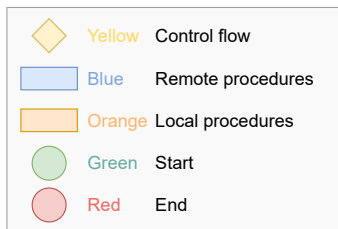
When a new job is submitted on the Computing Platform, ByteNite sets up a pipeline between the user and the grid. First, the Feeder builds the framework of the scheduling logic for that specific job, and the Reward System estimates its cost. Hence, the job starts and the Job Upload API streams the input data to the Partitioner, creating chunks on the fly and passing them on to the Feeder. The Feeder wraps them with an executable, forming tasks that are scheduled and sent to the grid. The distribution logic established by the Feeder’s scheduling algorithm guarantees the abstraction of the scheduling from the actual delivery so that the process is completely automated and reliable. In particular, the algorithm of the Feeder enforces the concept of “first come, first served”, so that no data chunk needs to wait for a specific device to show up, but every chunk is appended to global queues from which the next available device can download it (see §3.3.5). Every device competes in the grid to process as many tasks as it’s eligible for, and its only assignment is to tune in with ByteNite’s server to wait for new tasks in the global queues, to process them and upload back the results (Figure 3). The grid responds asynchronously, sending back processed tasks from multiple devices. Several measures are adopted to guarantee hassle-free continuation of the processing (see §3.3.6) when node failures or delays are encountered. In any case, the workflow continues up to the moment when all tasks have been successfully processed, retrieved, and validated. Finally, the Assembler quickly rebuilds the integral output using

indexes contained in tasks' metadata and uploads it to a Cloud bucket immediately available to the user.

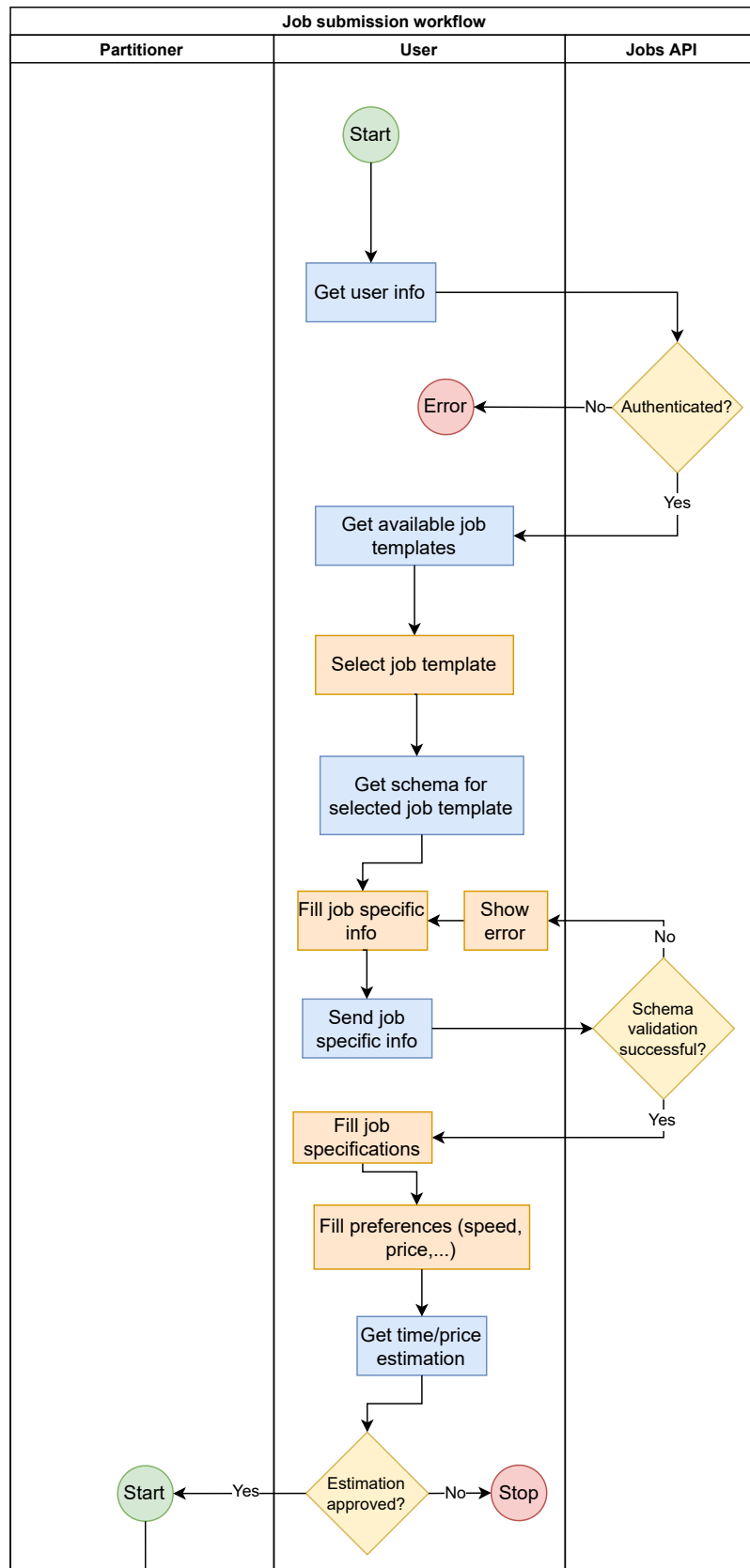
All data that goes through the Core System is temporarily stored and released as soon as a job is completed, except for the final output which can be stored in a Cloud bucket for 24 hours. Because neither the Partitioner nor any other services are tasked with the heavy lifting of data processing, ByteNite removes the need to maintain a high-capacity infrastructure. At the same time, ByteNite can control the inflow and outflow efficiently and insure the integrity and security of data processing.

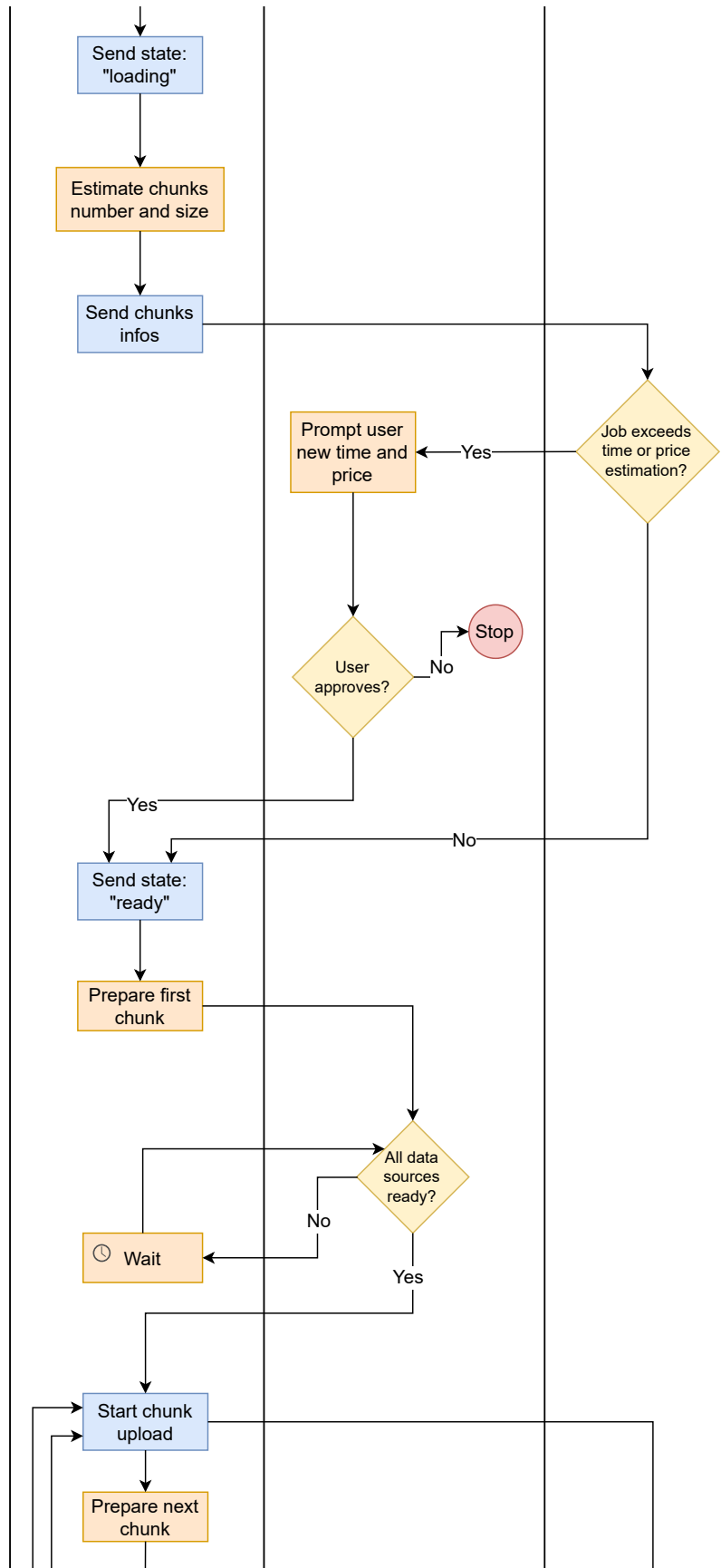
In the following diagrams, I will detail two key workflows: the job submission workflow running across the Computing App and the Core System, and the task processing running on the Worker App.

Legend:



(continues on next page)





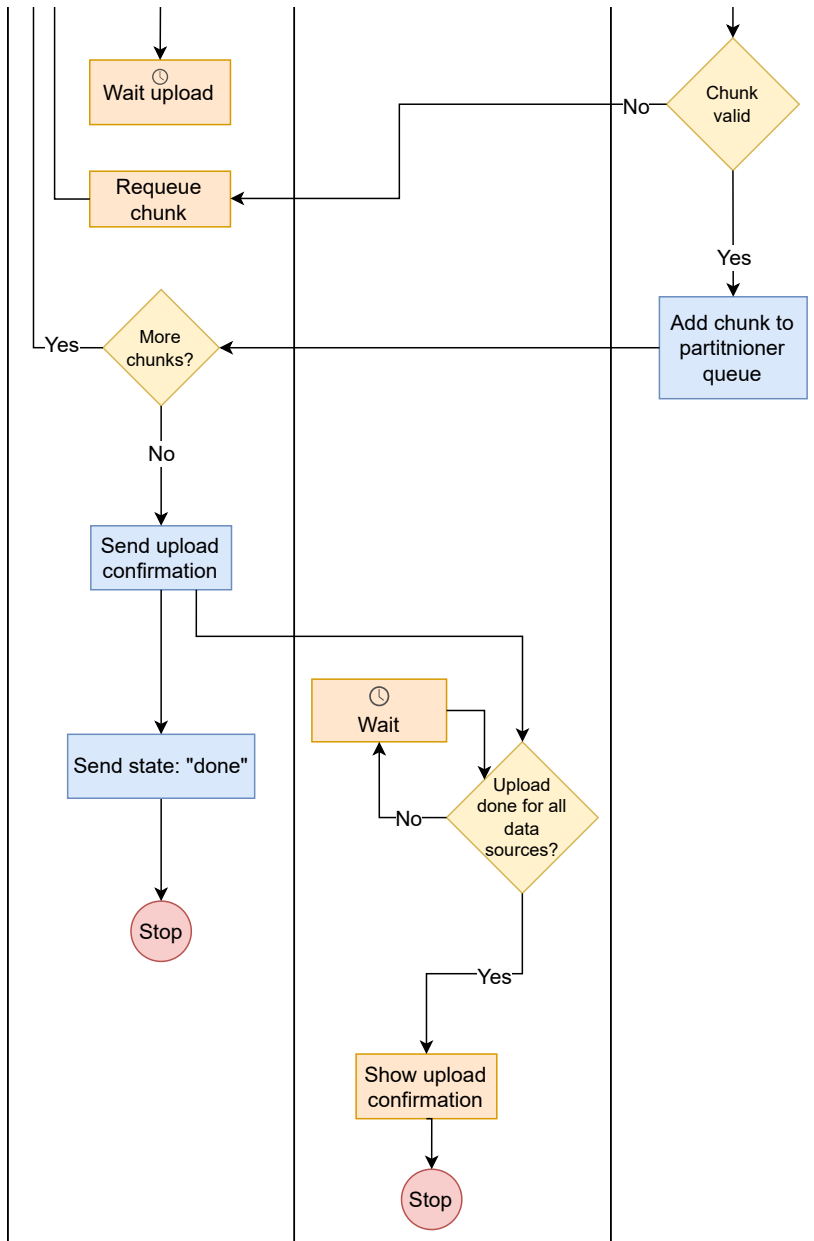
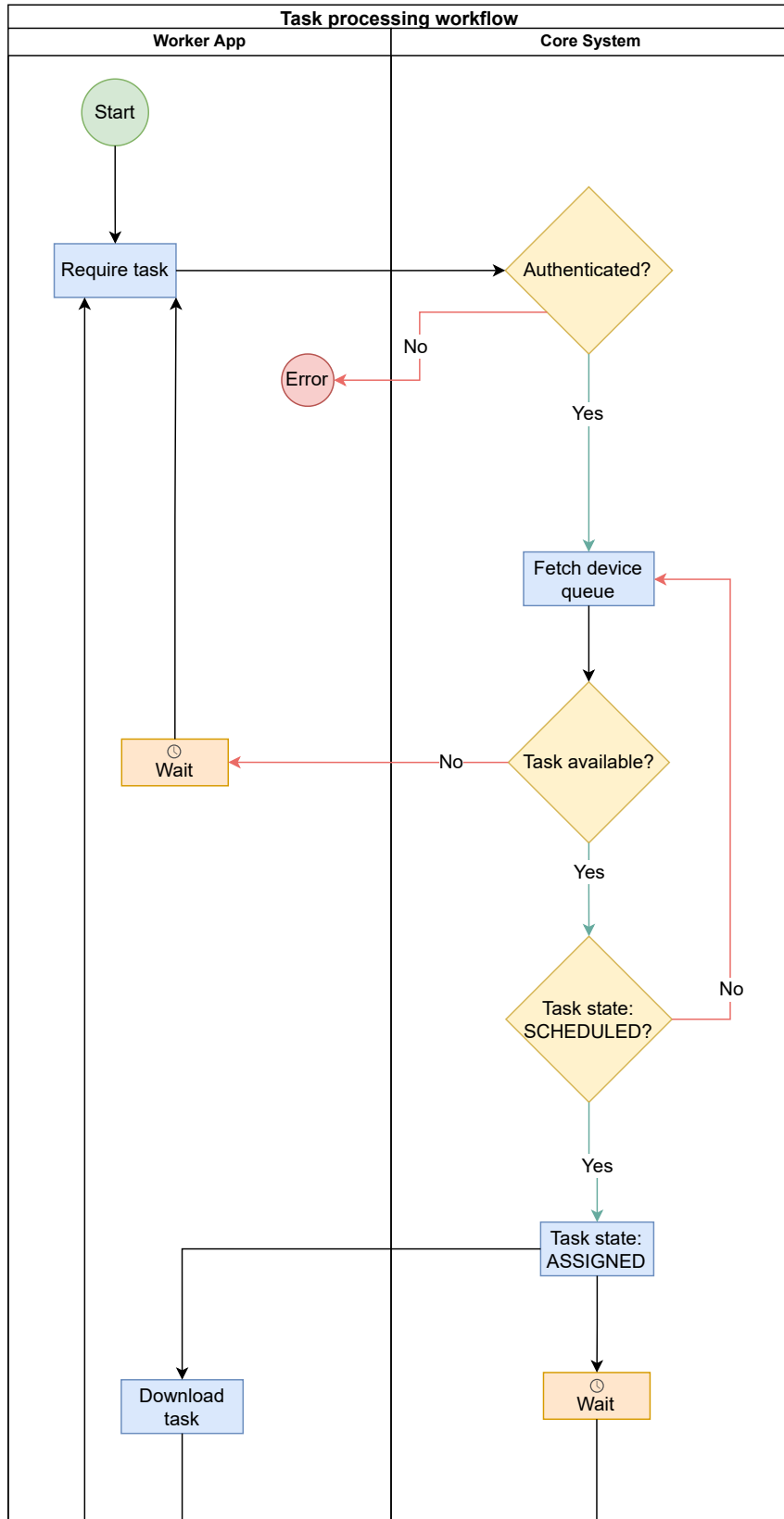


Figure 3 – The job submission workflow, running across the partitioner, the Computing Platform, and the Jobs API.



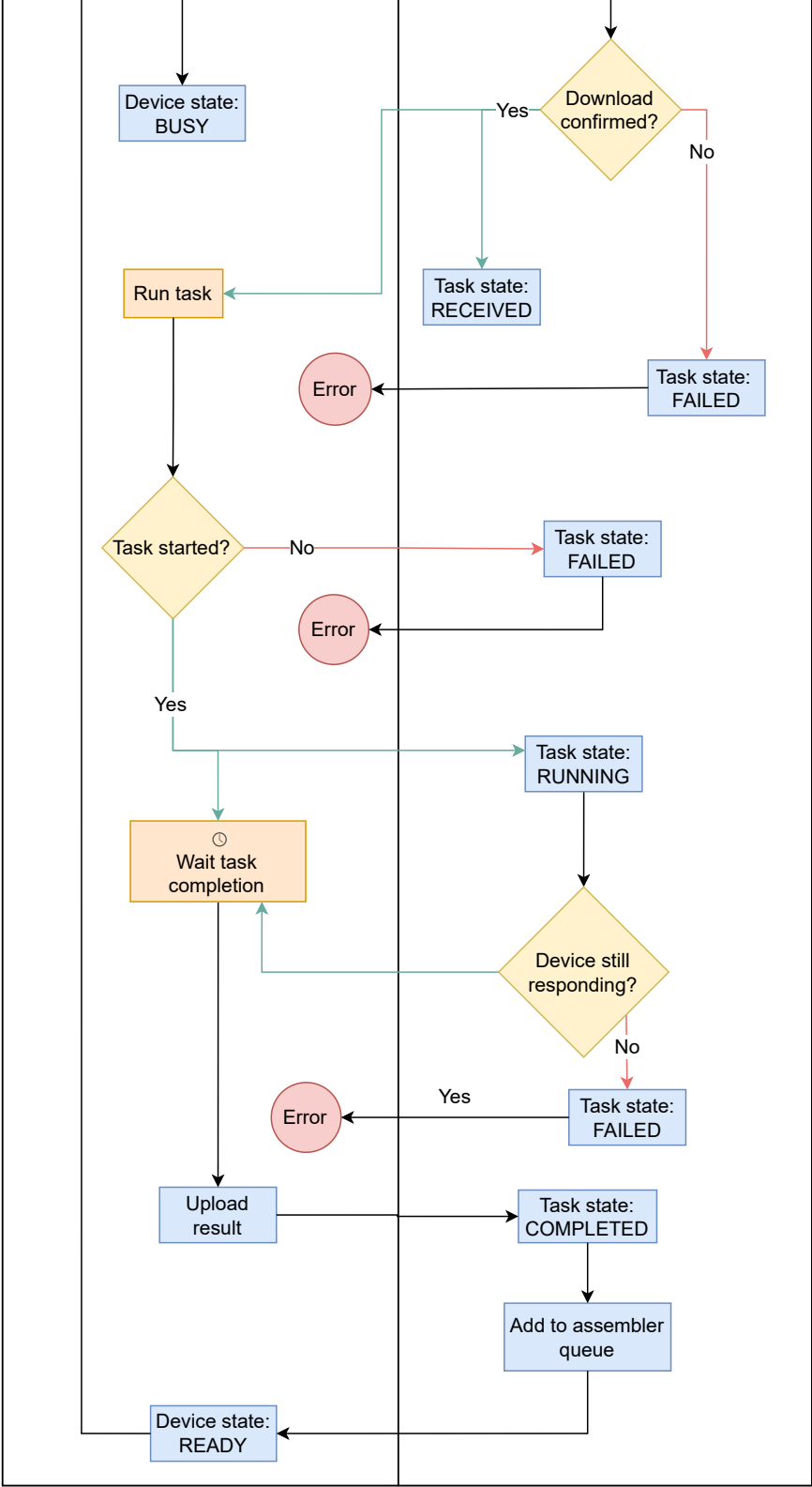


Figure 4 – The task processing workflow allows the worker devices to ask for tasks from the Core System and process them.

3.3. Business Logic

ByteNite’s innovative data partitioning and task distribution mechanism is ruled by a patent-pending algorithm that implements and optimizes the business logic underlying the operational decisions, like which and how many tasks are assigned to each device for any given job. The algorithm is constantly updated by our team; however, I’d like to provide an overview of what the algorithm will grow into as development proceeds and more data points are collected. For ByteNite to fulfill its attributes of Availability, Agility, and Speed, it is critical that this mechanism will eventually work as efficiently as, for instance, Google’s PageRank. That’s why I will focus on the business advantages of specific flows and structures composing the algorithm, and I shall name the device ordering protagonist of this section “ByteRank”.

The first domain of information referred to as “grid indexes”, is described in §3.3.2. The grid indexes are job-independent parameters that optimize knowledge about devices by giving them a score, a ranking, and several flags. They are built by a function in the Feeder that continuously collects devices’ information and runs formulas. Next, I describe the scheduling algorithm in §3.3.4, which combines the grid indexes with the job’s information to produce a bespoke partitioning pattern and a distribution rule. Finally, I cover how said distribution rule is turned into an actual workflow (§3.3.5) and how failures and delays are handled (§3.3.6).

3.3.1. Global Inputs

ByteNite must have access to all the relevant information about any device’s state, any new job’s metadata, and any possible user’s preferences to understand how to efficiently assign tasks to the grid. This is a crucial responsibility that drives the performance of ByteNite’s computing service. In this paragraph, I shall list the types of information that ByteNite uses to feed the algorithm, which I generally refer to as “global inputs”.

3.3.1.1. Grid State

ByteNite collects devices’ static and dynamic data, (e.g., manufacturer specs and memory or network usage), and stores it in the Firebase database, which is updated every 30 seconds. Here is a list of monitored parameters on each worker device through the Worker App:

<i>Parameter</i>	<i>Update</i>	<i>Example</i>
Operating system	One time	Android, Windows
Processor’s manufacturer, family, and model	One time	Intel i7-6700K
CPU cores	One time	4
GPU model	One time	NVidia GeForce 8600M GT
RAM capacity	One time	8 GB
Total storage capacity	One time	256 GB

<i>Parameter</i>	<i>Update</i>	<i>Example</i>
Worker App's online state	5s	Online / Offline
Network connection type	30s	Wi-Fi, 4G
Network connection speed	30s	46 Mbps
Network connection latency	30s	28 ms
IP address	30s	192.0.2.1, 2001:db8:0:1234:0:567:8:1
Available (unused) RAM	30s	2.23 GB
Available (unused) storage	30s	145 GB
Battery percentage (if applicable)	30s	86%
Charging state	30s	Plugged in, unplugged
Core temperature	30s	73 F

In addition, ByteNite needs to know if the worker has specific usage limitations that are configurable from the Worker App:

<i>Parameter</i>	<i>Update</i>	<i>Example</i>
Scheduled availability	Upon worker's request	SAT-SUN 01:00AM-12:00PM MON-FRI 00:00AM-07:00AM
Allowed network types	Upon worker's request	Only Wi-Fi
Data traffic threshold	Upon worker's request	Uncapped

Tables 2 & 3 – *Parameters of the grid state. The first table shows the parameters the worker devices automatically send to ByteNite according to a schedule ("Update" column). The second table shows the worker's preferences which are set once by the worker and updated at will.*

This collection of information, plus other Firebase data like login and usage history, is associated with every device and jointly constitutes the "grid state". The state is used to rank the devices according to their general availability and predisposition to perform computing tasks at any given time, as explained in §3.3.2.

3.3.1.2. Job Specification

The kind of information that a new job generates can be varied and unstructured. In the first place, it depends on the type of computing application, e.g., video encoding, text processing, or graphics rendering. Every supported application running on ByteNite must have a corresponding implementation on the Computing Platform (job templates and parameters schema), in the Core System (a bespoke partitioning tool), and on the Worker App (the actual distributed script that runs the application). Consequently, different

applications require different classes of data. In general, all these applications share two categories of input information besides the raw data:

- Metadata. May include any of the following:

<i>Metadata</i>	<i>Examples</i>
Input format	MP4, MKV, MOV OBJ, STL, STEP, 3DS
Input size	3.04 GB, 12 Mbps
Input length	30 min 124 documents 92,000 frames
Other type-specific data	60 fps, 1920x1080p, AVC

- Job parameters. They are input by the user and used to configure the application and the job submission workflow. I'll provide below a list of parameters for the Video Encoding application, with possible generalizations:

<i>Application</i>	<i>Parameter</i>	<i>Examples</i>
(any)	Input source	A link to a user-owned cloud bucket
(any)	Output destination	A bucket name, access key, and secret key, to save the output on a user-owned bucket
Video Encoding	Output format	MP4, MKV, WebM
Video Encoding	Output aspect	1280x720p, crop top & bottom by 10%, rotate 180° clockwise
Video Encoding	Output video specs	H.265, 30fps, VBR 700k
Video Encoding	Output audio specs	AAC, 2.0 channels, 48 kHz, 192kbps
Video Encoding / Computer Vision	Output video filters	Color correction, object detection

Tables 4 & 5 – The information contained in a job specification. Table 4 refers to all the metadata attached to a job or, equivalently, the description of the input data. Table 5 refers to the user-defined job parameters used to configure the applications.

3.3.1.3. Job Preferences

Before submitting a job, the user can specify other parameters, which allow to tweak the job execution's speed, trustworthiness, and other factors. While these parameters don't affect the specification of a job – meaning that the applications don't use them – they affect the scheduling algorithm run by the partitioner and the feeder by reflecting the user's

preferences on how to execute the job. Below are the user's preference parameters and their summary descriptions:

<i>User's preference</i>	<i>Description</i>	<i>Values</i>	<i>Implementation</i>
ByteLevel	Sets the overall performance, in terms of speed and priority, of the job	<ul style="list-style-type: none"> <i>low</i> <i>balanced (default)</i> <i>high</i> 	The scheduling algorithm takes this choice into account to select specific capacity pools and a different task distribution that allow the job to get done faster (High) or slower (Low)
Trusted processing	Ensures that the customer's data is processed only on verified nodes	<ul style="list-style-type: none"> <i>on</i> <i>off (default)</i> 	The scheduling algorithm filters out all the unverified devices in the grid when creating the target subset
Allowed regions	Specifies a set of geographical areas where ByteNite can send the user's job to be processed by the local workers	<ul style="list-style-type: none"> <i>any (default)</i> <i>Americas</i> <i>Europe</i> <i>Asia and Pacific</i> 	The scheduling algorithm filters out all the devices outside the allowed regions when creating the target subset
Preferred hardware	Specifies a range of processors, machines, or device configurations to be used to process the job	<ul style="list-style-type: none"> <i>desktop computers</i> <i>GPUs</i> <i>Wi-Fi-connected devices</i> 	The scheduling algorithm enforces the processing on the specified devices, though the target subset filter and a modification of the computing application

Table 6 – The job preferences expressed by the user. These inputs will drive the scheduling algorithm towards decisions that tweak the execution and performance of a job.

3.3.2. Grid Indexes

A set of summary metrics, or indexes, are computed and constantly updated to summarize and sort all the data points of the grid state and reflect every device's availability and potential. I shall describe their composition and usage in the following paragraphs.

3.3.2.1. Capacity Score

The capacity score is a number expressing every device's availability and computing capacity on an increasing scale. It is computed through several additional indexes that quantify the device's immutable computing capacity (e.g. processor type), and its dynamic availability (e.g. the network speed over time). The score is updated anytime one of its inputs changes, according to the update schedule presented in [Tables 2 & 3](#). The capacity score is intended to prize more performing processors, strong network connections, and high RAM and CPU time availability.

The capacity score (C) is the sum of the additional scores presented in [Table 7](#):

$$C = \text{processor score} + \text{network category score} + \text{network speed score} + \\ + \text{network stability score} + \text{RAM availability score} + \text{CPU availability score}$$

Equation 1 – The capacity score's formula. Refer to Table 4 for the composition of each term.

Every term of the sum is already weighted to reflect the importance of its contribution to the capacity score (see column "Range" below). The minimum value of the capacity score is 0, and the maximum is 2700. Here is the description and composition of each term of the sum:

Score	Description	Default	Formula	Range
Processor Score	Ranks all known processors' based on their performance	250	External benchmarking	1 – 500
Network Category Score	Prizes more stable network connection types up to 300 points	100	<ul style="list-style-type: none"> • Fiber/Cable/DSL = 300 • Starlink = 100 • Mobile & other = 0 	0 – 300
Network Speed Score	Prizes higher network speeds up to 1000 points, with 70% of the prize concentrated between 0Mbps and 200Mbps	100	$\text{score} = \frac{1}{2}L \left[(1 - e^{-kx}) + (1 - e^{-k(uy)}) \right]$ $x = \text{download speed}$ $y = \text{upload speed}$ $L = 1000$ $k = 0.00601986$ $u = 1.66666667$	0 – 1000
Network Stability Score	Prizes less variable network speeds over the last 10min interval up to 300 points	100	$\text{score} = c \cdot \max \left\{ 0, 1 - \frac{\sigma}{\mu} \right\}$ $c = 300$ <p>where μ and σ are, respectively, the mean and the standard deviation of a sample of network speeds over the past 10 minutes</p>	0 – 300
RAM Availability Score	Prizes a RAM availability higher than 256MB up to 300 points, with two thirds of the prize between 256MB and 1024MB	100	$\text{score} = \max \{ 0, L(1 - e^{-k(x-x_0)}) \}$ $x = \text{available RAM in MB}$ $L = 300$ $k = 0.00143048$ $x_0 = 256$	0 – 300
CPU Availability Score	Prizes a CPU availability higher than 10% up to 300 points, with two thirds of the prize between 10% and 50%, and max prize at 100%	100	$\text{score} = \max \{ 0, ax^2 + bx + c \}$ $x = \text{percentage of available CPU}$ $a = -0.03333333$ $b = 7$ $c = -66.66666667$	0 – 300

Table 7 – The list of scores that compose the capacity score.

3.3.2.2. Fault Rate

The fault rate examines the past behavior of a device by counting the number of times it has failed or delayed the execution of tasks. Unlike the capacity score, which estimates the performance of a device, the fault rate forecasts the outcome of a task, which may be unrelated to the capacity score of the device that ran it. Suppose, for instance, that a very well-equipped Mac with a capacity score of 2400 has installed an application that conflicts with ByteNite’s process, causing the App to crash and the tasks to fail. Before the engineers manually exclude the device from the grid and send a notification to the worker, the Mac would keep getting a high number of tasks, and there would be no action to prevent further task failures.

With the accountability of the past behavior expressed by the fault rate and the threshold flags discussed later in §3.3.2.4, it is possible to progressively discredit or rapidly exclude faulty devices from the task execution and trigger automatic quarantines and technical checks.

There are three types of faulty operation on a task: failure, delay, or incorrectness. The reader can explore about this topic further in §3.3.6. The fault rate takes the last 20 faulty operations that are attributable to the worker device and produces a number between 0 and 1 which is the average faulty operation occurrence. Here are the inputs used in the fault rate’s formula further down:

Input	Description	Values	Example
\underline{F}	Failure state of the past 20 processed tasks	1 = failed 0 = succeeded	$\underline{F} = (0, 0, 1, 0, 1, 0, 0, \dots)$
\underline{D}	Delay state of the past 20 processed tasks	1 = delayed 0 = on time	$\underline{D} = (1, 1, 1, 0, 0, 0, 0, \dots)$
\underline{I}	Incorrectness state of the past 20 processed tasks	1 = incorrect 0 = correct	$\underline{I} = (1, 0, 0, 0, 0, 0, 0, \dots)$

Table 8 – The fault rate’s inputs. Every vector records the last 20 outcomes of tasks run on a device.

Hence, the formula for the fault rate (R) is:

$$R = \frac{1}{3} \left[\mu(\underline{F}) + \mu(\underline{D}) + \mu(\underline{I}) \right]$$

Equation 2 – The fault rate’s formula. Refer to Table 5 for the definition of the variables \underline{F} , \underline{D} , and \underline{I} . μ is the sample mean.

As explained later, the fault rate lowers a device’s ByteRank, but only up to the point where the faulty operations are frequent enough to trigger the temporary exclusion of the device from the ranking. A script regenerates a clean fault rate by resetting the vectors \underline{F} , \underline{D} , and \underline{I} to 0 after the device has been reviewed and any problem causing the repeated failures has been fixed.

3.3.2.3. Repechage Lottery

The repechage lottery is a mechanism to grant low-ranked devices a second chance to prove they're worth a higher ByteRank and receive more tasks. The entire grid is enrolled in the lottery, and the winners get their ByteRank increased by 100 positions relative to the ranking they would normally have without the lottery. The lottery is run daily, and the rank increase is valid for 24 hours.

The lottery is based on a random draw where every device has a different probability of winning, and the outcome is positive or negative. The probability of winning (l) is computed from the hours elapsed since the last processing of a task and is maximum at 12h elapsed, where it attains the value of 5%. While I'll discuss the implementation of the repechage lottery in §3.3.3, I provide here the formula for the probability of winning, which is a grid index:

h = hours elapsed since the last task processing

$c = 0.00256564$

$\beta = 0.16666667$

$$l = c \cdot h^2 e^{-\beta h}$$

Equation 3 – Formula of the probability of winning the repechage lottery for a generic device.

This lottery has the effect of randomly pushing inactive devices up in the rankin. It tries to prevent high-performing devices from absorbing all ByteNite's tasks and rewards. Since we expect no more than 5 devices out of 100 to be boosted daily, this alteration doesn't disrupt ByteNite's efficient ordering.

3.3.2.4. Threshold Flags

Although numerical indexes are helpful in discerning and ranking devices accurately, it is critical for ByteNite to identify which devices shouldn't be enrolled in the scheduling algorithm at all and pass on that information to the feeder. Hence, I introduce indicators, or "flags", that signal when certain grid values have been exceeded or switched. Each flag is part of a device state as recorded in the Core System and helps make scheduling decisions accordingly. Flags can capture the most obvious state changes, like the Worker App online state, or alarming values, like a low battery level. The following table shows the flags and their threshold or rules:

Flag	Description	Threshold value
Disconnected	Indicates that a device is offline	<i>worker App's online state = offline</i>
Repeated failures	Indicates that a device has a too high fault rate	$R > 0.5$
Low battery	Indicates that a device has a too low battery level	<i>battery percentage < 10 %</i>
High temperature	Indicates that a device has a too high core temperature	<i>core temperature > 120° F</i>

Table 9 – The threshold flags are raised on exceeding or matching specific threshold values of a device state.

3.3.3. ByteRank

The grid indexes defined above are further summarized in an ultimate ranking that I call ByteRank. The ByteRank sorts all the devices in ByteNite's grid according to their overall expected capability, intended as a measure that encompasses all the perspectives analyzed so far, from the expected performance expressed by the capacity score to the predicted probability of failure of the fault rate. A device's ByteRank ultimately answers the question, "how much potential does this device currently have relative to the other devices in the grid?". In fact, it is also used to rule out flagged devices by setting their value to 0. That's also why it must be deemed a primary reference for any scheduling choices.

The ByteRank is computed iteratively starting from the threshold flags, then the capacity score, and finally the fault rate. It outputs a unique positive number for every device representing the rank, or "0" if the device is excluded from the rank. The procedure that builds the ByteRank is described below.

Lottery script – runs once a day:

INPUT

$(l_1, l_2, l_3, \dots, l_M)$ the vector of the devices' probabilities of winning the repechage lottery

START

*Generate a random variable from the Bernoulli distribution $B(l_i)$ for every device i in the grid.
The outcome is a vector of 0s and 1s, where "1" means the device has won the lottery*

END

Main script – runs every 30 seconds or less:

START

Build a linked list B containing the IDs of the m devices that are not flagged

↓

Order the list according to the capacity score of the devices in descending order

↓

Scan the list from bottom to top and, at iteration i , push the $m - i$ device by $\lfloor iR \rfloor$ positions towards the bottom, where R is the fault rate of such device (assuming the list numbering starts at 0)

↓

Push each device that won the lottery by 100 positions toward the top of the list

↓

The ByteRank of a device is its position in the list B , or "0" if it is not found in the list

END

Algorithm 1 – *The creation of the ByteRank.*

Hence, the ByteRank of each device is the index of the ordered list representing all the currently available devices ranked according to their capacity score, fault rate, threshold flags, and outcome of the repechage lottery. The top-ranked device has a ByteRank of 1, and the bottom-ranked device has a ByteRank of n (a number lower or equal to the total number of worker devices N). I will refer to the devices with a ByteRank different than 0 as the “active devices”, and n as the size of the active grid.

In the following paragraph, I’ll discuss how the ByteRank and the other grid indexes turned into actionable parameters that feed the scheduling decisions any time a new job is submitted.

3.3.4. Scheduling Algorithm

The scheduling algorithm represents the core of the business logic of ByteNite. It decides how to prepare and assign tasks to a subset of the active grid. It comes into action at any job submission, and, in a matter of milliseconds, outputs a chunk distribution and a correspondence between device pools and chunk sizes to instruct the partitioning and distribution pipeline (see §3.3.5). Throughout the following paragraphs, I will describe how the scheduling algorithm works; I will refer to “the job” as a generic job submission and to “the ByteRank” and “the grid indexes” as a snapshot of the ByteRank and the grid indexes taken at the instant that job is submitted.

When the job’s metadata is acquired, the feeder reads the grid state and the grid indexes and, together with the partitioner, selects a subset of candidate devices through two operations: the election of eligible devices and the creation of capacity pools.

3.3.4.1. Eligibility

Before the task creation and assignment logic are built, the active devices are filtered to meet job-specific requirements. App version, device location, or other information about a device might prevent it from supporting the job, depending on the job’s compatibility and how stringent the user’s preferences are. To be deemed eligible for the job, a device must meet the following requirements, when applicable:

- Match the job’s hardware restrictions, like processor and GPU type;
- Match the job’s data exporting restrictions, or be located in the geographical area allowed by the job;
- Be a Trusted Node, if required by the job;
- Have installed a version of the App compatible with the job;

The filtered list of active devices created in this step is called the “target subset” of the job and is passed on to the next stage of the algorithm.

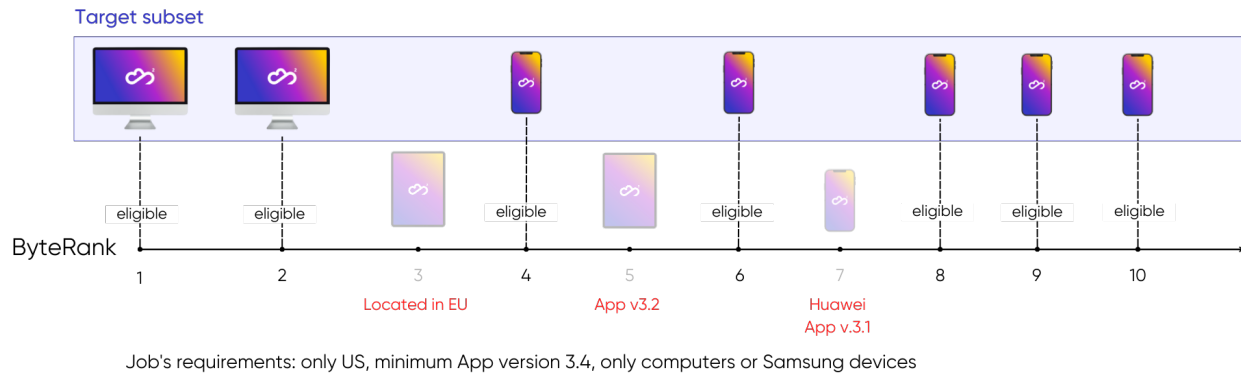


Figure 5 – The election of eligible devices and the creation of the target subset. Eligibility criteria include job-specific requirements and user preferences.

3.3.4.2. Capacity Pools

The clean and filtered Byte-ordered list of devices – the target subset – represents the ultimate array of candidate devices for the current job. At this point, if the workload were simply randomly assigned to the target subset, without prioritizing top-ranked devices over low-ranked ones, the utility of the ByteRank would cease. Considering that the target subset can potentially contain thousands of devices, and their actual performance can vary from a few seconds to a few minutes per task, it would be impossible to predict the final throughput if every device could fetch any tasks indiscriminately. Therefore, the question that motivates this paragraph is, “what is an efficient way to assign tasks to the target subset while maintaining a constant throughput all over it?”.

I propose a model consisting of the creation of device pools (“capacity pools”) working jointly with the task queuing system described in the distribution process (§3.3.5) to solve the workload assignment challenge proposed above. The capacity pools model is based on grouping the devices into clusters, or pools, according to their position in the ByteRank and their capacity score. The cumulative capacity score in every pool must be higher than a threshold, and the size of each pool cannot be smaller than another one. Therefore, the pools end up being balanced in capacity and size. The procedure to build the pools is as follows:

INPUT

- The target subset $B(j)$ of the ByteRank for job j*
- The capacity scores of the grid (C_1, C_2, \dots, C_M)*
- The threshold minimum value C_{min} of the cumulative capacity score for a pool*
- The minimum pool size s_{min}*

START

- Create a pool as a virtual collection of device IDs.*
- Place the device at the bottom of the target subset (lowest ByteRank) in the pool*



Scan the target subset from bottom to top.

At each iteration i :

- if the last updated pool has a cumulative capacity score lower than C_{min} or a size lower than s_{min} merge the device i with the last updated pool
- else, create a new pool with device i and create a link from it to the previous pool



The resulting list of pools containing devices IDs are the capacity pools

END

Algorithm 2 – The creation of capacity pools

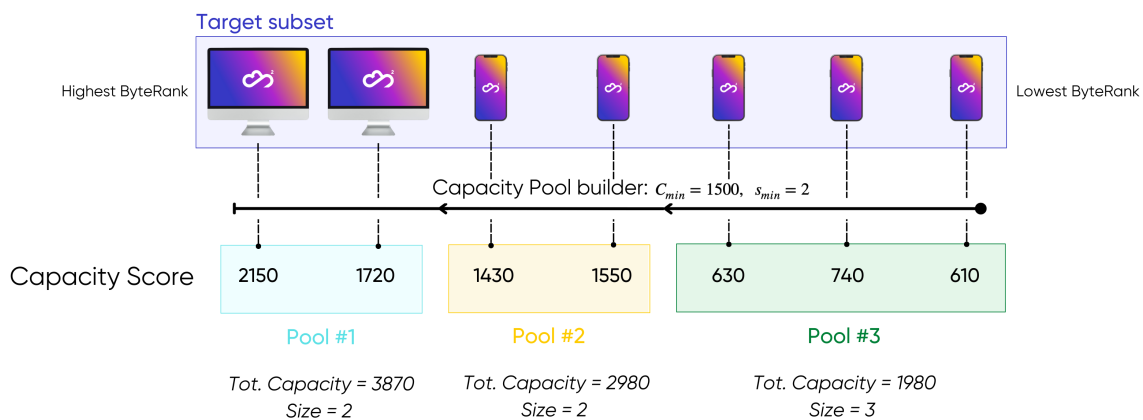


Figure 6 – The creation of capacity pools (visual). The target subset's devices are grouped starting from the low-ranked, with the rule to create a new pool when either a minimum capacity or size has been reached. Later, pools are numbered in the reverse order, i.e. from top-ranked to bottom-ranked.

Although I won't cover how C_{min} and s_{min} are set in this version of the paper, I'd like to note that, given the median capacity score of the target subset C_{me} , it is suitable that $C_{min} < C_{me} \cdot s_{min}$ to avoid creating pools that are too large or too small. While C_{min} helps to create well-balanced pools where the overall Capacity is similar, s_{min} is a measure against the production of small high-capacity clusters that absorb too many tasks and no longer benefit from parallelization.

As explained in the distribution process, an equal number of tasks will be assigned to each capacity pool, and every device will be able to process only the tasks assigned to its pool with a queue inheritance mechanism.

The introduction of the capacity pools activates the potential of the grid indexes in connection with the execution of a job. Devices with lower capacity scores will tend to overcrowd the pools they have been assigned to, while top-ranked devices will be in moderately crowded pools. Since the tasks are distributed in equal numbers to each pool, the top-ranked pools will get more tasks in proportion to their size, which is reasonable as they get their work done faster. As a result, the capacity pools end up balancing the

throughput vs. workload ratio over the grid and optimizing the speed of the overall execution. Even though this might seem sufficient, there are several other measures to make the task-processing workflow more efficient, which I will discuss in the following paragraphs.

3.3.5. Distribution Process

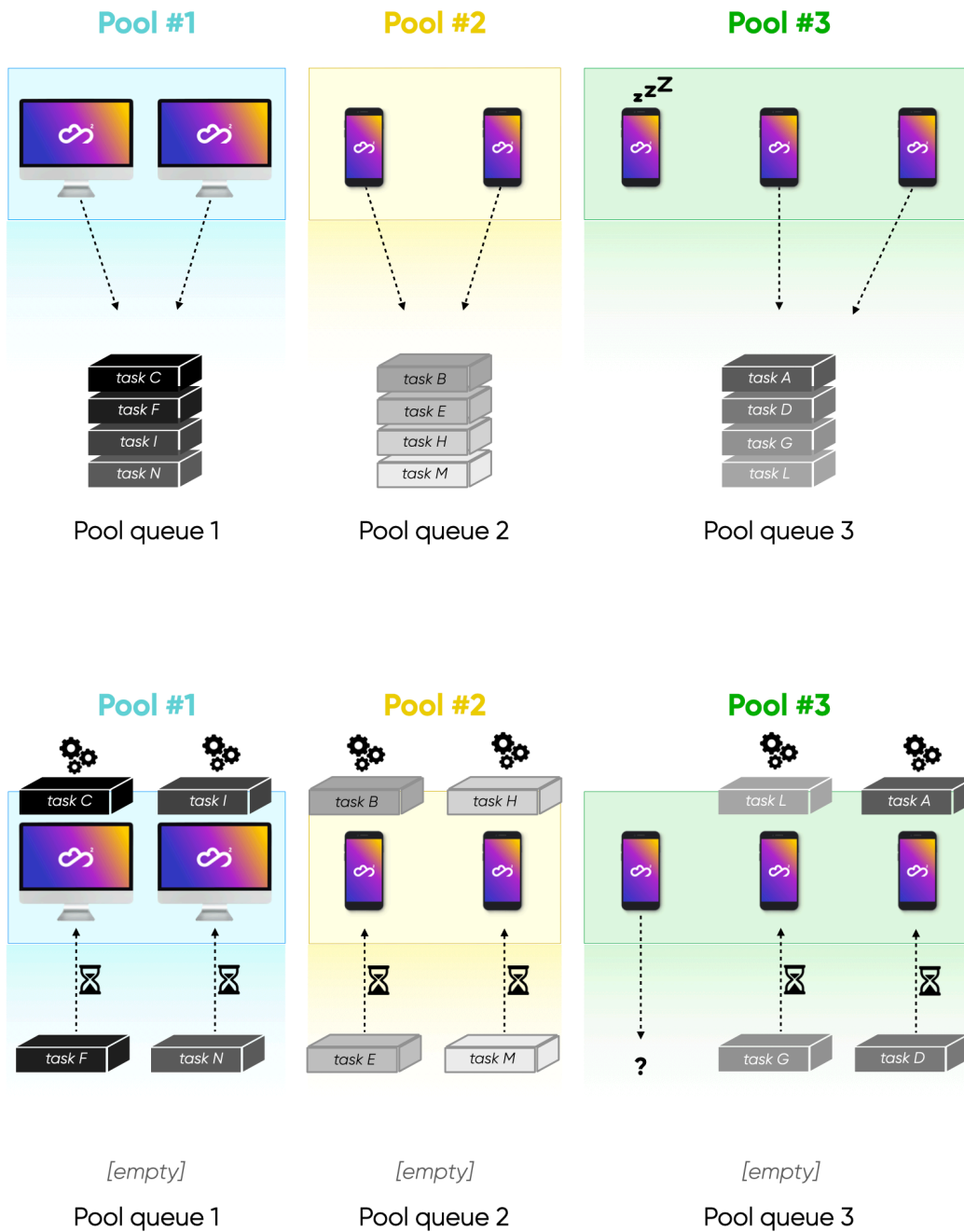
The distribution process is a workflow run across the partitioner and the feeder in charge of managing the distribution and supervising the processing of tasks over the grid for any given job. It is activated with the job submission workflow and continues until all the tasks have been retrieved, verified, and transmitted to the assembler.

First, the partitioner creates a stream that uploads the user's data in chunks. The chunks' size and distribution depend on the data and an application-specific partitioning rule. The partitioner sends every chunk to the feeder on the fly. The feeder accordingly forms tasks by putting together a chunk, the computing application, and all the metadata needed by the worker device. On the Core System, every task has a record with a unique identifier and information such as the ByteChip bounty, the expected processing time, the target capacity pool, and the application parameters.

Next, the process creates a specific queue of tasks for every pool (the pool queues), filling each with the same number of newly produced tasks, ready to be deployed (see [Figure 7](#)). At this point, the feeder does not notify devices about the assignment of a task, but it is up to each device to query ByteNite's server and pick the next available task in its pool queue. Each device can keep two tasks simultaneously, one being processed while the other is downloaded, to absorb the time for data transfers as much as possible ([Figure 8](#)).

When a task is completed, it is uploaded to the Core System according to the workflow examined in [Figure 4](#). The validator will check the task's integrity, like size, length, or data type. If the task has a replica, the last completed replica will be used by the validator to check the correctness of the computation. When any of these validations have been passed, the task is moved to the assembler, which will progressively rebuild the output and terminate as soon as the last task has arrived.

The reverse assignment enforced by the pool queues guarantees a "first come, first served" concept that stimulates competition in each pool about who is the fastest to query a new task, complete it, and query more. It all benefits the overall performance. However, the performance could still be endangered by processing failures or delays. In the following paragraph, I will explain the threats posed by such shortcomings, and discuss both a-priori and a-posteriori measures to contain them.



Figures 7 & 8 – The pool queues of tasks before and after being deployed. Tasks are equally distributed to the queues as soon as they are produced. The worker devices periodically query the server requiring new tasks: the fastest to reach the Tasks API will get the next task in the queue, while the slow or temporarily unavailable ones might miss the opportunity to process.

3.3.6. Fault Tolerance

Whenever a worker device doesn't respond as expected during the processing of a task, the overall performance of the job is endangered. To illustrate, suppose that a job requires the execution of 12 tasks, as in [Figures 7 & 8](#), and that every task takes a maximum of 40 seconds to be downloaded, processed, and uploaded by either device. If six devices are processing the tasks as in the picture, and no failure occurs, we should expect all tasks to come back in 80 seconds, and the job to be finished in a little more than that. However, suppose only one device, the device processing tasks H and M, fails. In that case, the system has to reschedule the tasks on a new machine. Depending on when the failure occurred, it can take up to 40 additional seconds to get the job done, or a 50% increase in time. This is assuming that the rescheduled tasks don't fail or that figure can even grow further.

Generalizing the previous simple argument, the exposure to frequent and stochastic faults in the grid can determine severe delays and nullify ByteNite's intelligent scheduling efforts if not dealt with properly. Although there is no ultimate solution to prevent or fix such delays, some strategies help mitigate them.

Before exploring such strategies, I shall briefly overview the possible causes of task failure. There are three types of faulty operations on a task:

- *Raising of an execution or transfer error.*
This happens when a task can't be fully downloaded or uploaded by the assigned device or can't be successfully processed on it and returns an error to the Core System.
- *Delayed task or unresponsive device.*
In this case, the "delayed" task's state is not updated by the device, but by the Core System after multiple attempts to connect to the device.
- *Incorrect or partial result.*
A task fails when it outputs a partial or incorrect result. To establish a benchmark for any result that is suspiciously incorrect, besides meeting expected or desired output metadata (size, length, etc.), ByteNite can run the same task on a trusted machine and compare results or use a replica processed by another device.

In addition, for each of these failures, ByteNite should establish whether the fault is due to the worker or ByteNite itself in order to compute a fair fault rate (see [§3.3.2.2](#)). ByteNite might inadvertently match tasks with unsupported devices or wrong App versions, or deploy inherently faulty applications. Despite being very careful about sending unverified executables or deploying untested releases, these situations are possible. That's why ByteNite runs double-check tests on trusted machines, and only when none of these tests fail is the failure blamed on the worker.

Two strategies help minimize the delays occurring when tasks unexpectedly fail: one operates a-priori, or before any failure occurs, and the other a-posteriori, or as a consequence of the failure.

3.3.6.1. Replication Strategy (A-Priori)

As a preliminary measure to fight task failures and consequent performance losses, ByteNite introduces a standard in distributed computing: a replication strategy. The reader can find a full-length analysis of the statistical framework behind this paragraph in the Appendix.

The rationale behind the replication strategy is that since only one result is needed by each task to deem it completed, we could be confident that at least one task wouldn't fail if we sent the same task to several different devices. To reword this in technical terms, introducing task replicas helps reduce the joint probability of failure for a set of identical tasks.

The strategy is based on the definition of a parameter, r , called the replication factor, which sets the number of replicas of the tasks. If m is the number of unique tasks that a job generates, then the total number of tasks (original and replicas) will be:

$$M = \lfloor r \cdot m \rfloor$$

Task replicas are not treated differently from their originators. In fact, every replica is deployed to the grid as a standard task and follows the same rules of the scheduling algorithm. The only way it is linked to the original task is a simple task identifier and a replica count.

The replication factor can be a decimal number. In such cases, only a portion of the tasks are replicated or further replicated (e.g., if $r = 1.5$, half of the tasks are replicated once). To decide which tasks are replicated and which are not, I propose a model, better detailed in the Appendix, that sorts the tasks according to the fault rate of the expected assigned devices. Hence, the tasks expected to be assigned to devices with a higher fault rate are replicated first. In practice, the mathematical model shows that the maximum benefit occurs when r assuming integer numbers, e.g., 2 or 3, and the benefit increases with r .

By having multiple devices work on the same tasks, the replication strategy not only has the effect of containing joint failures, but it allows us to exploit the best performance over every replica set. As a matter of fact, the partitioner uses the first returned result of each replica set to build the final output and uses the possible other returned replicas for validation. However, setting a higher replication factor implies using more computing power and possibly involving more devices than would normally take part in a job's processing. ByteNite burdens the increase in resource usage demanded by replication partly on the user – by increasing the job's price for higher replication factors – and partly on the workers – by splitting the bounty across different replicas.

Although a theoretical approach provides a benchmark for weighing scheduling decisions, it doesn't explain how to enforce a timely communication of a task's failure, nor a strategy to solve the corner case where all replicas fail. A practical, a-posteriori strategy is thereby needed.

3.3.6.2. Fault Response Strategy (A-Posteriori)

Before a task result is spontaneously sent from a device to ByteNite and is accordingly tagged as completed or failed, some time passes. Usually, a task doesn't last more than one or a few minutes, as ByteNite's philosophy is based on a real-time, disposable supply of computing resources. During that time, there is no assurance that the task will be completed in the near future, nor that it will be completed at all. That's where the constant monitoring of the active devices, also anticipated in [Figure 4](#), comes into play. The following diagram explains the feeder's workflow to monitor the active devices and promptly take action when a failure or delay is encountered. It is based on the definition of two timeouts T_1 and T_2 , and it ends by switching the task's state into one of "COMPLETED", "FAILED", or "STALE". Whenever the task has one of the last two states, the feeder looks for replicas of that task: if none is in

process or completed, the ultimate response to the fault is to reschedule the task on a fast, trusted device.

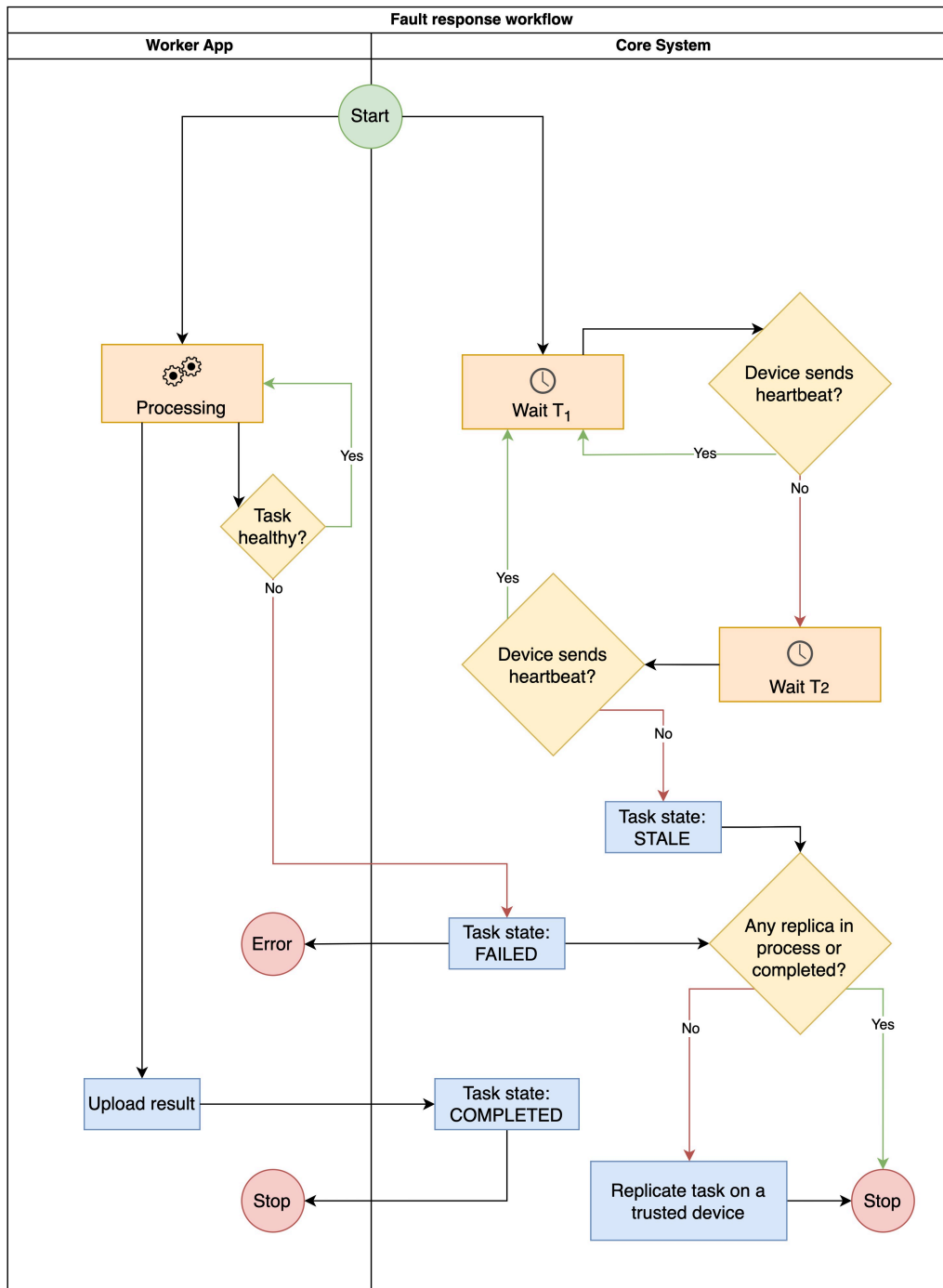


Figure 9 – The fault response workflow. The term "heartbeat" refers to a simple periodic signal generated by the worker device, communicating its regular operation to the Core System.

Bibliography

- [1] B. A. Sosinsky, *Cloud Computing Bible*, 1st edition ed. (Bible v.757). Indianapolis, Ind: Wiley (in English), 2011.
- [2] M. Bote-Lorenzo, Y. Dimitriadis, and E. Gómez-Sánchez, "Grid Characteristics and Uses: A Grid Definition," 2004, pp. 291-298.
- [3] L. Ferreira and O. International Business Machines Corporation. *International Technical Support, Introduction to grid computing with Globus*, 2nd ed. (IBM redbooks). San Jose, CA: IBM Corp., International Technical Support Organization (in English), 2003.
- [4] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," *Cloud Computing and Grid Computing 360-Degree Compared*, vol. 5, 01/31 2009, doi: 10.1109/GCE.2008.4738445.
- [5] R. Buyya. "Grid Computing Info Centre (GRID Infoware)." <http://www.gridcomputing.com/> (accessed 2022).
- [6] R. Buyya and K. Bubendorfer, *Market-oriented grid and utility computing (Wiley series on parallel and distributed computing)*. Hoboken, N.J: John Wiley & Sons (in English), 2010.
- [7] P. B. Charlie Catlett, Dane Skow and Ian Foster, "Creating and Operating National-Scale Cyberinfrastructure Services," *CTWatch Quarterly*, vol. 2, May 2006. [Online]. Available: <https://icl.utk.edu/ctwatch/quarterly/print.php%3Fp=35.html>.
- [8] T. G. Alliance. "The Globus Toolkit." <http://toolkit.globus.org/> (accessed 2022).
- [9] I. Foster and C. Kesselman, "The Globus project: a status report," *Future Generation Computer Systems*, vol. 15, no. 5, pp. 607-621, 1999/10/01/ 1999, doi: [https://doi.org/10.1016/S0167-739X\(99\)00013-8](https://doi.org/10.1016/S0167-739X(99)00013-8).
- [10] "SETI@Home." University of California, Berkeley. <https://setiathome.berkeley.edu/> (accessed 2022).
- [11] "BOINC." University of California, Berkeley. <https://boinc.berkeley.edu/> (accessed 2022).
- [12] D. P. Anderson, "BOINC: a system for public-resource computing and storage," in *Fifth IEEE/ACM International Workshop on Grid Computing*, 8-8 Nov. 2004 2004, pp. 4-10, doi: 10.1109/GRID.2004.14.
- [13] "List of volunteer computing projects," in *Wikipedia*, 2022 ed.
- [14] "Einstein@Home." <https://einsteinathome.org/> (accessed 2022).
- [15] "World Community Grid (WCG)." Krembil Research Institute. <https://www.worldcommunitygrid.org/> (accessed 2022).
- [16] "climateprediction.net." University of Oxford. <https://www.climateprediction.net/> (accessed 2022).

-
- [17] "distributed.net." <https://www.distributed.net/> (accessed 2022).
- [18] "HTCondor." HTCondor Team, Center for High Throughput Computing, Computer Sciences Department, University of Wisconsin-Madison, WI. <https://htcondor.org/> (accessed 2022).
- [19] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the Condor experience: Research Articles," *Concurrency - Practice and Experience*, vol. 17, pp. 323-356, 02/01 2005, doi: 10.1002/cpe.938.
- [20] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and V. S. Pande, "Folding@home: Lessons from eight years of volunteer distributed computing," in 2009 IEEE International Symposium on Parallel & Distributed Processing, 23-29 May 2009 2009, pp. 1-8, doi: 10.1109/IPDPS.2009.5160922.
- [21] "Livepeer." <https://livepeer.org/> (accessed 2022).
- [22] D. Petkanics and E. Tang, "Livepeer whitepaper," Technical report, Livepeer, 2018.
- [23] "Theta Network." <https://www.thetatoken.org/> (accessed 2022).
- [24] "Sweatcoin." <https://sweatco.in/> (accessed 2022).
- [25] "Pi White Paper," SocialChain, Inc., 2019. Accessed: December 2022. [Online]. Available: <https://minepi.com/white-paper>
- [26] "The Golem Project," Golem Factory GmbH, 2016. Accessed: 2022. [Online]. Available: <https://whitepaper.io/document/21/golem-whitepaper>
- [27] "CUDOS." <https://www.cudos.org/> (accessed 2022).
- [28] "CUDOS White Paper," Cudos Limited, 2021. Accessed: 2022. [Online]. Available: <https://www.cudos.org/wp-content/uploads/2021/11/cudos-white-paper.pdf>
- [29] G. Fedak, W. Bendella, and E. Alves, "iExec - Blockchain-Based Decentralized Cloud Computing (Whitepaper)," iExec, 2018. [Online]. Available: https://iex.ec/wp-content/uploads/2022/09/iexec_whitepaper.pdf
- [30] R. B. Uriarte and R. DeNicola, "Blockchain-Based Decentralized Cloud/Fog Solutions: Challenges, Opportunities, and Standards," *IEEE communications standards magazine*, vol. 2, no. 3, pp. 22-28, 2018, doi: 10.1109/MCOMSTD.2018.1800020.
- [31] "SONM Supercomputer Organized by Network Mining," SONM, 2017. Accessed: 2022. [Online]. Available: <https://whitepaper.io/document/326/sonm-whitepaper>
- [32] A. Chandra, J. Weissman, and B. Heintz, "Decentralized Edge Clouds," *IEEE Internet Computing*, vol. 17, no. 5, pp. 70-73, 2013, doi: 10.1109/MIC.2013.93.
- [33] M. Arslan, I. Singh, S. Singh, H. Madhyastha, K. Sundaresan, and S. Krishnamurthy, "Computing while charging: Building a distributed computing infrastructure using smartphones." 2012, pp. 193-204.
- [34] H. Ba, W. Heinzelman, C.-A. Janssen, and J. Shi, "Mobile computing -A green computing resource." 2013.
-

- [35] C. Borcea, X. Ding, N. Gehani, R. Curtmola, M. A. Khan, and H. Debnath, "Avatar: Mobile Distributed Computing in the Cloud," in 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, 30 March–3 April 2015 2015, pp. 151–156, doi: 10.1109/MobileCloud.2015.22.
- [36] D. Díaz-Sánchez, A. Marín-López, F. Almenares, R. Sanchez, and P. Cabarcos, Flexible Computing for personal electronic devices. 2013, pp. 212–213.
- [37] Z. Dong et al., "REPC: Reliable and efficient participatory computing for mobile devices," 2014 11th Annual IEEE International Conference on Sensing, Communication, and Networking, SECON 2014, pp. 257–265, 12/16 2014, doi: 10.1109/SAHCN.2014.6990361.
- [38] I. Foster, "What is the Grid? A Three Point Checklist," GRID today, vol. 1, pp. 32–36, 01/01 2002.
- [39] G. Massari, M. Zanella, and W. Fornaciari, "Towards Distributed Mobile Computing," in 2016 Mobile System Technologies Workshop (MST), 23–23 Sept. 2016 2016, pp. 29–35, doi: 10.1109/MST.2016.13.
- [40] I. Raicu et al., "Toward loosely coupled programming on petascale systems," in SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, 15–21 Nov. 2008 2008, pp. 1–12, doi: 10.1109/SC.2008.5219768.
- [41] I. Raicu, Y. Zhao, I. Foster, and A. Szalay, "Accelerating Large-scale Data Exploration through Data Diffusion," CoRR, vol. abs/0808.3546, 01/01 2008, doi: 10.1145/1383519.1383521.
- [42] W. Cai, Z. Wang, J. B. Ernst, Z. Hong, C. Feng, and V. C. M. Leung, "Decentralized Applications: The Blockchain-Empowered Software System," IEEE Access, vol. 6, pp. 53019–53033, 2018, doi: 10.1109/ACCESS.2018.2870644.
- [43] Z. Hong, Z. Wang, W. Cai, and V. C. M. Leung, "Connectivity-Aware Task Outsourcing and Scheduling in D2D Networks," in 2017 26th International Conference on Computer Communication and Networks (ICCCN), 31 July–3 Aug. 2017 2017, pp. 1–9, doi: 10.1109/ICCCN.2017.8038386.

Mathematical Appendix

I. Replication Strategy

In this section, I describe a statistical framework for the replication strategy introduced in §3.3.6.1. I detail the assumptions based on the available information of the grid state, set certain goals of the analysis, and finally lay down the calculations that bring to the solution.

In the scenario of assigning m tasks to the computational grid, let's define two strategies: a simple strategy S , where tasks aren't replicated, and a replication strategy R , where $M = \lfloor r \cdot m \rfloor$ devices are involved in the computation of m unique tasks.

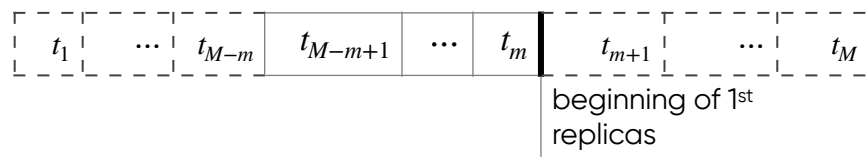
Depending on $r \geq 1$, the tasks can be replicated a different number of times. Suppose that we can schedule each task to a specific device in advance. Suppose further that it is possible to estimate the probability that device i will fail, independently on the task, and call it p_i . A possible estimate of p_i is the fault rate of a device introduced in §3.3.2.2. Now, let's order the devices $i = 1, \dots, M$ from the one with the highest probability of failure, or the least reliable, p_1 , to the one with the lowest probability of failure, or the most reliable, p_M . Calling t_i the task executed on device i , the replica assignment in R works as follows:

- $1 \leq r < 2$

The last $M - m$ devices are assigned the same tasks of the first $M - m$.

This implies $t_i = t_{i+m}$ for $i = 1, \dots, M - m$

There is no replica of $2m - M$ tasks and there are 2 replicas of $M - m = \lfloor (r - 1)m \rfloor$ tasks.

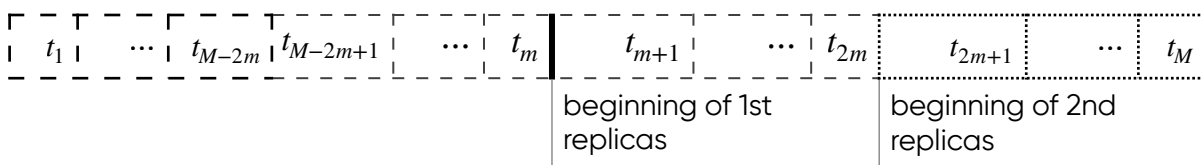


- $2 \leq r < 3$

The first m devices are assigned the same task of the second m devices. The last $M - 2m$ devices are assigned the same tasks of the first $M - 2m$ devices.

This implies $t_i = t_{i+m}$ for $i = 1, \dots, m$ and in addition $t_i = t_{i+2m}$ for $i = 1, \dots, M - 2m$

There are 2 replicas of $3m - M$ tasks and there are 3 replicas of $M - 2n$ tasks.



- $a \leq r < a + 1$

There are a replicas of $(a + 1)m - M$ tasks and $a + 1$ replicas of $M - am$ tasks.

Let a set of independent variables $X_i \sim Be(p_i)$, $p_i \geq p_j$ for $i < j$ be given, each representing the outcome of a task on a device i , where $X_i = 1$ is the failure event occurring with probability p_i . Note that " $p_i \geq p_j$ " implies that the devices are ordered from the least reliable to the most reliable.

Problem

- Find the expression for the probability of joint failure of at least one set of replicas, in S and in R . I'll generally abbreviate this probability as PJF. Then, I'll call the expression found with the first strategy P_S , and the expression found with the replication strategy P_R .
- Find a rule to set a minimum r such that the probability of joint failure is below a selected threshold.

Solution in S

In the simple strategy S , the PJF is simply the probability of a single task failure:

$$P_S = \mathbb{P}\left(\sum_{i=1}^m X_i \geq 1\right) = 1 - \mathbb{P}\left(\bigcap_{i=1}^m \{X_i = 0\}\right) = 1 - \prod_{i=1}^m \mathbb{P}(X_i = 0) = 1 - \prod_{i=1}^m (1 - p_i)$$

Therefore, $1 - (1 - p_m)^m \leq P_S \leq 1 - (1 - p_1)^m$.

Furthermore, if we model $p_i = p \ \forall i$, then $P_S = 1 - (1 - p)^m$.

Solution in R , case $1 \leq r < 2$

Switching to the replication strategy R , the probability P_R that at least one set of replicas fails is no more equal to the probability of failure of at least one task, because the replica(s) of the failed task might be completed. In this case, the event we're looking for is when a task and all its replicas fail simultaneously.

In order to find P_R , we begin with the strategy $1 \leq r < 2$, where the total number of tasks M is such that $m \leq M < 2m$.

In this case, the PJF is equal to the probability that at least one couple of replicas fails or that at least one of the non-replicated tasks fails:

$$P_R = \mathbb{P}\left(\bigcup_{i=1}^{M-m} \{X_i = 1, X_{i+m} = 1\} \cup \bigcup_{j=M-m+1}^m \{X_j = 1\}\right) := \mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B)$$

Since the events in A and B involve variables in different ranges, which are independent, A and B are independent as well. Thus:

$$P_R = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A)\mathbb{P}(B) = \mathbb{P}(A)[1 - \mathbb{P}(B)] + \mathbb{P}(B)$$

$$\text{In addition, } P(A) = \mathbb{P}\left(\{X_i = 1, X_{i+m} = 1\}\right) = \mathbb{P}(\{X_i = 1\})\mathbb{P}(\{X_{i+m} = 1\}) = p_i p_{i+m} \quad \forall i$$

Let us then find $\mathbb{P}(A)$ and $\mathbb{P}(B)$ separately:

$$\begin{aligned} \mathbb{P}(A) &= \mathbb{P}\left(\bigcup_{i=1}^{M-m} \{X_i = 1, X_{i+m} = 1\}\right) \\ &= 1 - \mathbb{P}\left(\bigcap_{i=1}^{M-m} \{X_i = 1, X_{i+m} = 1\}^C\right) \\ &= 1 - \prod_{i=1}^{M-m} \mathbb{P}\left(\{X_i = 1, X_{i+m} = 1\}^C\right) \\ &= 1 - \prod_{i=1}^{M-m} \left[1 - \mathbb{P}\left(\{X_i = 1, X_{i+m} = 1\}\right)\right] \\ &= 1 - \prod_{i=1}^{M-m} (1 - p_i p_{i+m}) \\ \mathbb{P}(B) &= \mathbb{P}\left(\bigcup_{j=M-m+1}^m \{X_j = 1\}\right) = \mathbb{P}\left(\sum_{j=M-m+1}^m X_j \geq 1\right) = 1 - \prod_{j=M-m+1}^m (1 - p_j) \end{aligned}$$

Therefore,

$$P_R = \left[1 - \prod_{i=1}^{M-m} (1 - p_i p_{i+m})\right] \cdot \left[\prod_{j=M-m+1}^m (1 - p_j)\right] + 1 - \prod_{j=M-m+1}^m (1 - p_j)$$

When we model the probabilities $p_i = p \quad \forall i$, the latter has a simpler form:

$$\begin{aligned} P_R &= \left[1 - (1 - p^2)^{M-m}\right] \cdot [(1 - p)^{2m-M}] + 1 - (1 - p)^{2m-M} \\ &= (1 - p)^{2m-M} - (1 - p)^m (1 + p)^{M-m} + 1 - (1 - p)^{2m-M} \\ &= 1 - (1 - p)^m (1 + p)^{M-m} \end{aligned}$$

$$= 1 - (1 - p)^m (1 + p)^{\lfloor rm \rfloor - m}$$

What is, hence, the minimum $1 \leq r < 2$ that binds this probability to be lower than a pre-set threshold δ ?

$$P_R \leq \delta \Leftrightarrow 1 - (1 - p)^m (1 + p)^{\lfloor (r-1)m \rfloor} \leq \delta$$

$$\Leftrightarrow (1 + p)^{\lfloor (r-1)m \rfloor} \geq \frac{1 - \delta}{(1 - p)^m}$$

$$\Leftrightarrow \lfloor (r - 1)m \rfloor \geq \frac{\log\left(\frac{1 - \delta}{(1 - p)^m}\right)}{\log(1 + p)}$$

$$\Leftrightarrow (r - 1)m \geq \left\lceil \frac{\log\left(\frac{1 - \delta}{(1 - p)^m}\right)}{\log(1 + p)} \right\rceil$$

$$\Leftrightarrow r \geq 1 + \frac{1}{m} \cdot \left\lceil \frac{\log\left(\frac{1 - \delta}{(1 - p)^m}\right)}{\log(1 + p)} \right\rceil$$

The previous expression lets us set a desired amount of single replicas to constrain the PJF with δ , when the probabilities are uniform. However, with the assumption that a task cannot have more than one replica, not every desired probability threshold δ can be chosen, as the following line shows:

$$\delta > 1 - (1 - p)^m (1 + p)^{\lfloor (r-1)m \rfloor} > 1 - (1 - p)^m (1 + p)^m = 1 - (1 - p^2)^m$$

For $1 \leq r < 2$

It is clear from the previous consideration that, in order to decrease δ at will and be as confident as needed that a joint failure won't occur, we should introduce more than one replica. Before generalizing the formula to any number of replicas, let us see a couple of examples with this model.

- *Example 1*

Suppose we have estimated a uniform fault rate of $R = 0.05\%$ based on the history of similar tasks, where on average 1 out of 2000 trials on a device returned a negative response.

The partitioner has produced $m = 10,000$ unique tasks for j . Our customer wants its job processed quickly and demands a PJF of at most $\delta = 1\%$.

Question: How should we pick r ?

Solution: According to the 2-replica model, the minimum r to satisfy the customer's requirement is:

$$r = 1 + \frac{1}{10,000} \cdot \left[\frac{\log\left(\frac{1 - 0.01}{(1 - 0.0005)^{10,000}}\right)}{(1 + 0.0005)^{10,000}} \right] = 1.9985$$

In other words, $[rm] = 19,985$ tasks should be produced overall, 9,985 of which are replicas. The task assignment should then follow the rule described at the beginning of the section, where tasks assigned to the most faulty devices get a replica. However, in this case, only 15 tasks are left without a replica.

Question: How low can the customer go with δ , in the 1-replica model?

The customer can be satisfied down to $\delta = 1 - (1 - 0.0005^2)^{10,000} = 0.250\%$

- *Example 2*

The target subset of devices now has a more varied fault rate: old-generation smartphones have a $R = 4\%$; new-generation smartphones are way more reliable with $R = 0.1\%$; laptop and desktop computers are extremely stable and very rarely fail: $R = 0.005\%$; finally, a set of devices connected with specific networks often encounter communication issues and therefore fail to complete their task. For the last set of devices, it has been observed that they fail in $R = 30\%$ cases, with a standard deviation of $\sigma = 13.4\%$.

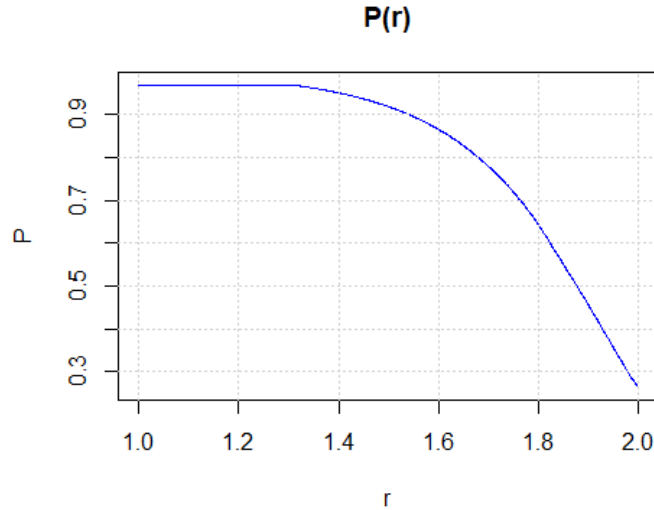
Suppose that the scheduling algorithm selects the following pools over the target subset: 700 old-generation smartphones, 6,800 new-generation smartphones, 1,600 laptop and desktop computers, and 900 devices with bad internet connection.

Question: How should we pick r to face the same demand of the customer ($\delta = 1\%$) for a job with $m = 5000$ unique tasks?

Solution: In this case, we must use the model with heterogeneous p_i . In addition, for illustrative purposes, we might either use the mean or a pseudo-random generator to simulate the p_i 's of the last set of devices. We opt for the second one.

Using a scientific calculator, it turns out that the customer cannot be satisfied in this

case, as 26,40% is the minimum probability achievable when $r = 2$. The profile of $P_R(r)$ function is the following, given the other parameters as above.



As it's clear from the last example, the lower bound for δ in the 1-replica model might be too high for certain probability configurations.

One can reasonably comment that the previous examples employed imaginary probabilities and that the real ones can be way lower.

Nonetheless, customers must be able to require any level of δ regardless of the state of the grid. Hence, it is not sufficient to stop our analysis at the level $1 \leq r < 2$.

Solution in R , general case

The results for the 1-replica model easily generalize to the case $a \leq r < a + 1$, $a \in \mathbb{N}$, where, following the task assignment previously defined, a total of $(a + 1)m - M$ tasks are replicated a times, and the remaining $M - am$ tasks have $a + 1$ replicas. Let's call the PJF P_R^a in the general case, and let's write and simplify its expression:

$$P_R^a = \mathbb{P} \left(\bigcup_{i=1}^{M-am} \bigcap_{k=1}^{a+1} \{X_{i+(k-1)m} = 1\} \cup \bigcup_{j=M-am+1}^m \bigcap_{k=1}^a \{X_{i+(k-1)m} = 1\} \right) :=$$

$$:= \mathbb{P}(A \cup B) = 1 - \mathbb{P}(A^C \cap B^C) = 1 - \mathbb{P}(A^C) \mathbb{P}(B^C)$$

where A and B are independent events (they involve mutually independent variables).

Let's find $\mathbb{P}(A^C)$ first:

$$\mathbb{P}(A^C) = \mathbb{P} \left[\bigcap_{i=1}^{M-am} \left(\bigcap_{k=1}^{a+1} \{X_{i+(k-1)m} = 1\} \right)^C \right] =$$

where $\left\{ \bigcap_{k=1}^{a+1} \{X_{i+(k-1)m} = 1\} \right\}_{i=1}^{M-am}$ is a collection of independent events, and so is the collection of their complementaries. Therefore:

$$\begin{aligned} &= \prod_{i=1}^{M-am} \mathbb{P} \left(\bigcap_{k=1}^{a+1} \{X_{i+(k-1)m} = 1\} \right)^C = \\ &= \prod_{i=1}^{M-am} \left[1 - \mathbb{P} \left(\bigcap_{k=1}^{a+1} \{X_{i+(k-1)m} = 1\} \right) \right] = \\ &= \prod_{i=1}^{M-am} \left(1 - \prod_{k=1}^{a+1} p_{i+(k-1)m} \right) \end{aligned}$$

In a similar way, it can be proven that:

$$\mathbb{P}(B^C) = \prod_{i=M-am+1}^m \left(1 - \prod_{k=1}^a p_{i+(k-1)m} \right)$$

The final formula is then:

$$\begin{aligned} P_R^a &= 1 - \prod_{i=1}^{M-am} \left(1 - \prod_{k=1}^{a+1} p_{i+(k-1)m} \right) \cdot \prod_{i=M-am+1}^m \left(1 - \prod_{k=1}^a p_{i+(k-1)m} \right) \\ & \quad a = \lfloor r \rfloor \quad , \quad M = \lfloor r \cdot m \rfloor \end{aligned}$$

Equation – The Probability of Joint Failure’s general formula, as a function of the replication coefficient r , the number of unique tasks m , and the probabilities of failure p_i , $i = 1, \dots, \lfloor r \cdot m \rfloor$ sorted in descending order.

In the homogeneous assumption $p_i = p$, and the general PJF boils down to:

$$\begin{aligned} P_R^a &= 1 - (1 - p^{a+1})^{M-am} \cdot (1 - p^a)^{(a+1)m-M} \\ &= 1 - \left(1 - p^{\lfloor r \rfloor + 1} \right)^{\lfloor rm \rfloor - \lfloor r \rfloor m} \cdot \left(1 - p^{\lfloor r \rfloor} \right)^{\lfloor r \rfloor - (\lfloor rm \rfloor - \lfloor r \rfloor m)} \end{aligned}$$

Finally, pooling all the models together, we can define a function of the replication coefficient r , by simply replacing a with $\lfloor r \rfloor$:

$$P_R(r) := P_R^{\lfloor r \rfloor}$$

The following example shows that multiple replicas can lead to adequate values of P_R even when the fault rates are very high.

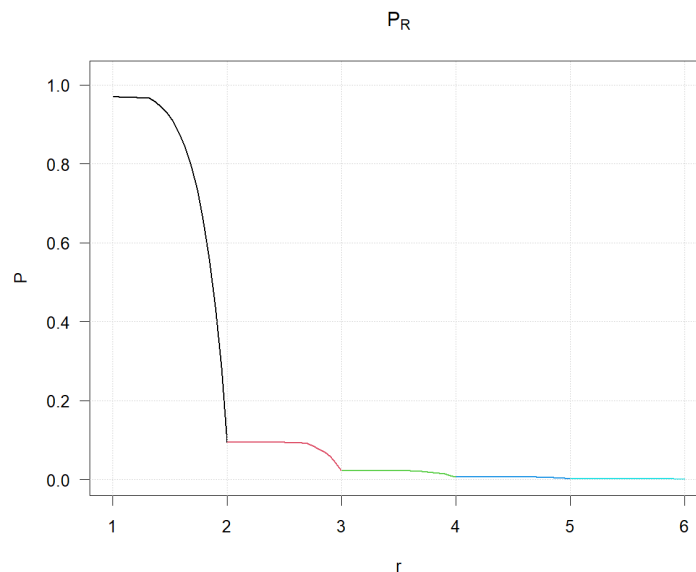
- *Example 3*

Suppose that the phone company XYZ, issuing a large share of the US mobile traffic, is experiencing a general breakdown and its users get randomly and independently disconnected from the network every now and then. As a consequence, the estimated fault rates of such devices are $R = 35\% \pm 12\%$.

Suppose further that the current grid is composed of the same devices as in Example 2, namely 10,000 devices with heterogeneous probabilities of failure ranging from 0.005% to $30\% \pm 13.4\%$. In addition, we have available an unlimited number of devices connected through the XYZ company's network.

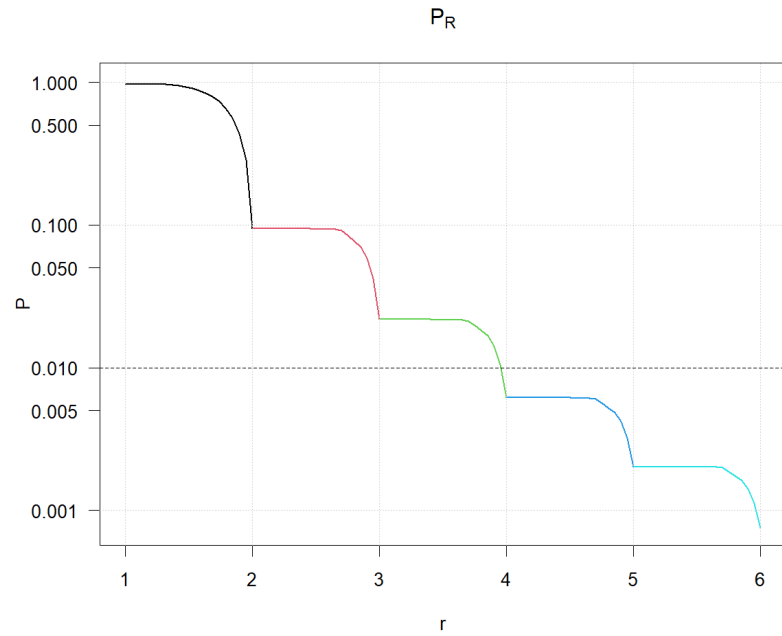
Question: In a job with $m = 5000$ unique tasks, what is the optimal replication coefficient $r \geq 1$ for which the constraint $\delta = 1\%$ is met?

Solution: When we allow multiple replicas, the probability function assumes the following profile:



From the picture, it is clear that the PJF is always a decreasing function of r . For this job in particular, the highest loss in the PJF is obtained when switching from no replicas ($r = 1$) to one replica ($r = 2$). However, in this case, we should go beyond two replicas in order to achieve a constraint of $\delta = 1\%$. We observe that the further we increase r , the lower the benefit from replication; indeed, the function has a negative exponential behavior, as it can be deduced from its expression.

The same plot in log-scale makes the point where the constraint is met more visible:



The optimal r , computed with an iterative non-linear equation solver, is around 3.956822.

Contact

<https://www.bytenite.com>

- Phone: [+1 415-723-2082](tel:+14157232082)
- Info & inquiries: info@bytenite.com
- CEO: fabio@bytenite.com
CTO: niccolo@bytenite.com

ByteNite Inc.
708 Long Bridge Street
Apt. 916
San Francisco, CA 94158
United States