future internet

MDPI

Perspective

Volunteer Down: How COVID-19 Created the Largest Idling Supercomputer on Earth

Nane Kratzke

Department for Electrical Engineering and Computer Science, Lübeck University of Applied Sciences, 23562 Lübeck, Germany; nane.kratzke@th-luebeck.de

Received: 25 May 2020; Accepted: 3 June 2020; Published: 6 June 2020

Abstract: From close to scratch, the COVID-19 pandemic created the largest volunteer supercomputer on earth. Sadly, processing resources assigned to the corresponding Folding@home project cannot be shared with other volunteer computing projects efficiently. Consequently, the largest supercomputer had significant idle times. This perspective paper investigates how the resource sharing of future volunteer computing projects could be improved. Notably, efficient resource sharing has been optimized throughout the last ten years in cloud computing. Therefore, this perspective paper reviews the current state of volunteer and cloud computing to analyze what both domains could learn from each other. It turns out that the disclosed resource sharing shortcomings of volunteer computing could be addressed by technologies that have been invented, optimized, and adapted for entirely different purposes by cloud-native companies like Uber, Airbnb, Google, or Facebook. Promising technologies might be containers, serverless architectures, image registries, distributed service registries, and all have one thing in common: They already exist and are all tried and tested in large web-scale deployments.

Keywords: volunteer computing; cloud computing; grid computing; HPC; supercomputing; microservice; nanoservice; container; cloud-native; serverless; platform; lessons-learned; COVID-19

1. Introduction

On 28 April 2020, Greg Bowman—director of the volunteer computing (VC) project Folding@home—posted this on Twitter https://twitter.com/drGregBowman/status/1255142727760543744:

> *@Folding@home is continuing its growth spurt! There are now 3.5 M devices participating, including 2.8 M CPUs (19 M cores!) and 700 K GPUs.*

According to Figure 1, the volunteer computing project Folding@home had even access to more computing power than all TOP-500 supercomputers together could provide! Just two months earlier, only 30,000 devices contributed to this project. Thus, what caused this tremendous increase in processing power? It was COVID-19 and the willingness of hundreds of thousands of "nerds" to fight COVID-19 by supporting Sars-CoV-2 computational bioscience research that was processed by the Folding@home project.

Our research is mainly dealing with cloud computing and corresponding software engineering and architecture questions. However, like many others, we surprisingly faced the COVID-19 shutdown impacts and decided on 23 March 2020 spontaneously to support Sars-CoV-2 computational bioscience-related research. Our lab set up two 32 CPU virtual machines from our virtualization cluster that is usually used for our cloud computing research and student projects.

Moreover, we called our students for support, and our students boarded almost immediately providing their most valuable assets: Gaming graphic cards. On 1 May 2020, we ranked on place 2145 of 252,771 teams worldwide (https://stats.foldingathome.org/team/245623). Thus, we were

suddenly among the top 1% of contributors as a very modest-sized University of Applied Sciences! Even the local newspaper reported about this. Furthermore, it is essential to mention that such kind of teams not only grew up in Lübeck, but this also happened all over the world. In total, the biggest supercomputer on earth emerged somehow by accident.

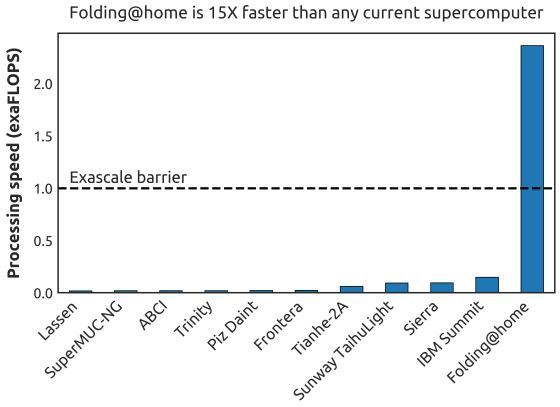


Figure 1. Processing speed posted by Folding@home project, 13 April 2020, https://twitter.com/foldingathome/status/1249778379634675712.

We were impressed by our students and by these numbers. Nevertheless, we also looked at the usage statistics of our provided machines gathered by our virtualization cluster. However, these data were so-so (see Figure 2). Our machines, although being configured to run at full power, had only usage rates between 40% or 50%. In other words, we had assigned two machines, but, on average, only one was used. At the end of March, this was even more severe. In that phase, many teams and individual contributors boarded and the Folding@home network grew massively. However, the effect was that our machines ran at ridiculous low usage rates. We first checked whether we had misconfigured the machines, but our machines were operating correctly. The processing pipelines were empty, and the master nodes needed hardware upgrades to handle all these new processing nodes. The control plane of the Folding@home project could not scale-out fast enough. In cloud computing, we would say that Folding@home was not elastic enough. It took all of April to make full use of the provided volunteer resources (see Figure 2).

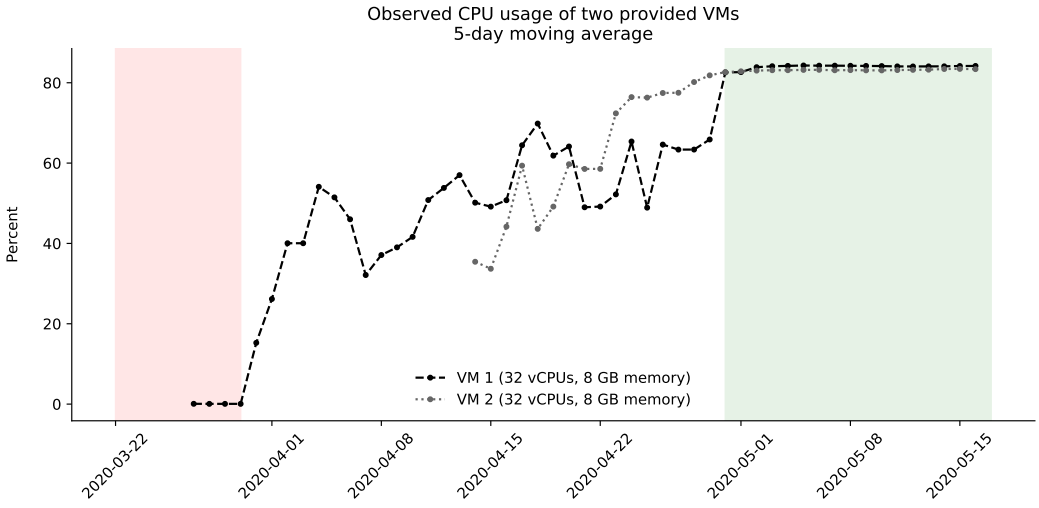


Figure 2. Example usage data of two virtual machines provided to the Folding@home project (compiled from our own data); red area: overload of master nodes, white area: recovering phase, green area: normal operation.

One obvious question arises here. Would it not be better in such a situation to use the amount of contributed nodes for other VC projects? If we cannot fight COVID-19, we might contribute to mathematical research, climate research, cancer research, or any other computationally intensive kind of research. The most of contributors will contribute their resources regardless of the specific aim of a VC project. Sadly, the reader will see throughout this paper that VC is not well prepared for shifting processing resources across projects that can lead to such undesirable usage scenarios shown in Figure 2.

Therefore, this paper deals with the question of how to improve the resource sharing of future VC projects. We have done some similar transfer research for another domain and asked what could be shifted auspiciously from the cloud computing domain to the simulation domain [1]. Because we derived some stupendous insights for the simulation domain, we will follow a quite similar methodology here (see Figure 3).

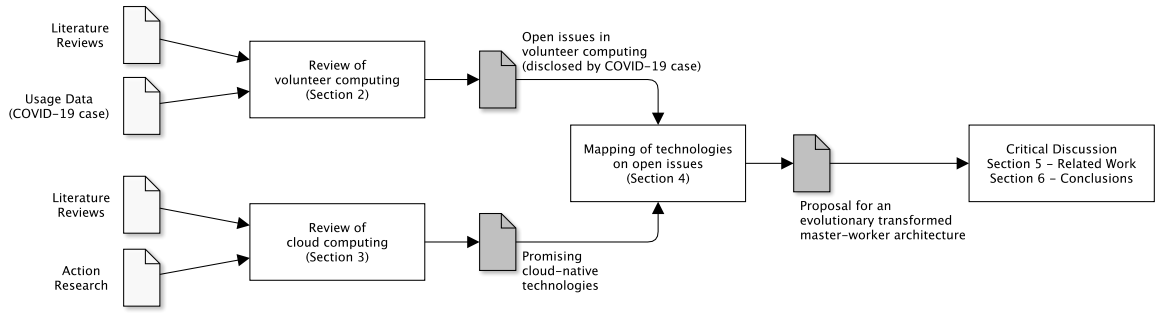


Figure 3. Research methodology.

The main contribution of this paper is to provide and explain engineering ideas and principles taken from the cloud computing domain that have been successfully invented, optimized, and successfully implemented by so-called cloud-native companies like Netflix, Uber, Airbnb, and many more. COVID-19 demonstrated that VC has some shortcomings regarding elasticity and efficient use of shared resources. Precisely, these two points have been optimized throughout the last ten years in cloud computing [2]. Cloud-native companies invented a lot of technology to make efficient use of shared resources elastically [3]. These companies were forced to invent these resource optimization technologies because it turned out that cloud computing can be costly if used inefficiently.

Thus, this paper strives to transfer some of these lessons learned [4] from the cloud computing domain to the VC domain that might be beneficial. Consequently, VC could be better prepared for flexible sharing of processing resources across different VC projects. VC could share more and not claim donated resources.

We start by reviewing the current state of VC in Section 2 doing the same for the cloud computing domain in Section 3. Section 4 will analyze what both domains could learn from each other and will derive some requirements and promising architectural opportunities for future VC projects. Section 5 will present the corresponding related work from the cloud and VC domain to provide interesting follow-up readings for the reader. We will conclude about our insights in Section 6 and forecast more standardized deployment units and more integrated but decentralized control-planes for VC.

2. Review on Volunteer Computing

Over the past 20 years, VC projects have contributed to hundreds of scientific research publications on a wide range of topics including climate change, clean energy, astrophysics, cancer research, malaria, earthquakes, mathematics, and the search for extraterrestrial intelligence. This research has been made possible by hundreds of thousands of people donating the unused processing power of their desktops, laptops, and even smartphones to researchers. Figure 4 shows the number and research disciplines of papers that are related to one of the most popular VC platforms (BOINC). It has been compiled

from a systematic review of VC projects [5] and the official list of publications by BOINC projects [6]. This chart might be not representative, but it gives a good impression on the kind of questions VC is used for.

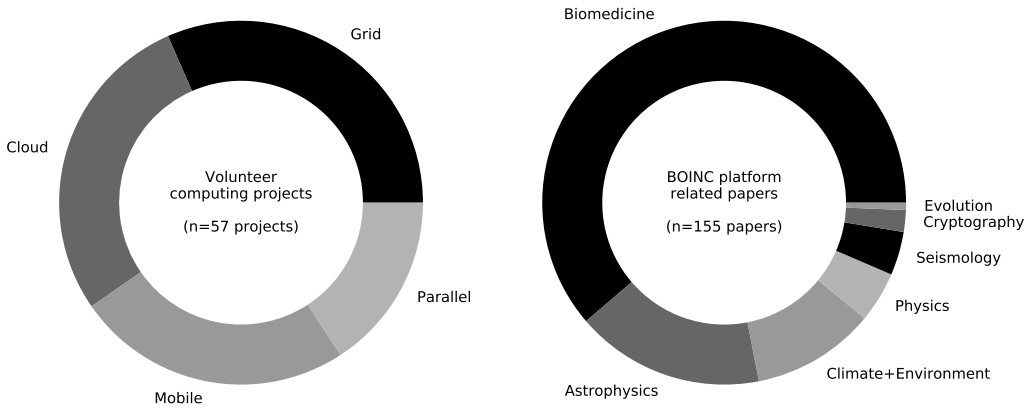


Figure 4. Categories and research disciplines of VC projects, data taken from [5].

2.1. Categories of Volunteer Computing

Figure 5 shows the publishing years of announcing VC solution proposal papers grouped by their major categories. We see that grid approaches affected VC, especially in the very beginning. However, since 2009, cloud as well as mobile approaches gained more and more influence on the community. The majority of VC systems are geared towards embarrassingly parallel tasks that need no or little communication (Bag of Tasks). According to [5], we classify such VC systems based on the computational environments that they are deployed in.

- **Volunteer Grid Computing** makes use of the aggregated computing resources of volunteer devices. *"Volunteer grids are one way of fulfilling the original goal of Grid Computing, where anyone can donate computing resources to the grid so that users can use it for their computational needs. Contrary to the traditional grid infrastructure, which needs a dedicated infrastructure to run on, volunteer grid runs on scavenging computing resources from desktop computers for computationally intensive applications."* [5] These kinds of systems form the majority of all VC approaches (see Figure 4).
- **Volunteer Cloud Computing** provides volunteer clouds as opportunistic cloud systems that run over donated quotas of resources of volunteer computers. Volunteer cloud systems come in different shapes, such as desktop clouds, peer-to-peer clouds, social clouds, volunteer storage cloud, and more [5]. The approach mimics the Cloud Computing service models (IaaS, PaaS, SaaS) without relying on centralized data centers that are operated by hyperscaling service providers like Amazon Web Services, Google, or Microsoft. These clouds are multipurpose and usually have no specific mission like volunteer grid computing projects that are focused on a specific research discipline or even a specific research question.
- **Mobile Volunteer Computing** makes use of advances in low-power consuming processors of portable computers such as tablets and smartphones that can handle computationally intensive applications. According to [5], nearly 50% of the worldwide population use smartphones and tablets—more than conventional Laptop and PCs. The increasing computing power, fast-growing number, and their power-efficient design make mobile devices interesting for distributed computing [7]. Consequently, many traditional VC systems are extended to include such devices. For example, BOINC provides an Android-based client (https://boinc.berkeley.edu/wiki/Android_FAQ).

- Most VC systems support embarrassingly parallel tasks that need no or little communication among the tasks [5]. However, **parallel VC** addresses use cases that need massive communication among the tasks, based on MPI, MapReduce, or other platforms.

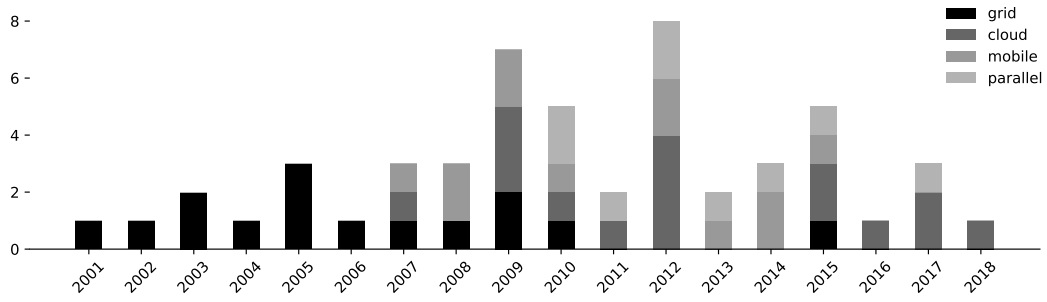


Figure 5. Publishing of VC solution proposals (project papers), data taken from [5].

Table 1 lists all projects and corresponding follow-up references for the information of the reader. We do not claim that this list is complete. This paper focuses mainly on Volunteer Grid Computing. However, it considers other VC computational environments if this seems appropriate.

Table 1. Overview of some VC projects of recent years, data taken from [5].

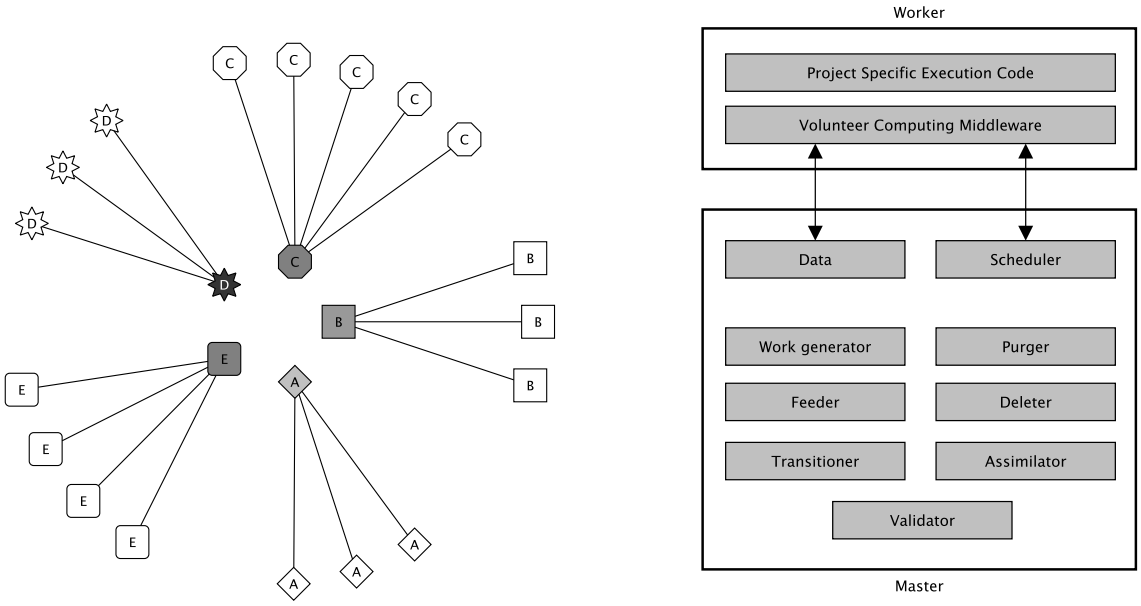
Category	Platforms and References (Ordered by Publication Date)
Volunteer grid computing	Condor [8], XtremWeb [9], SETI@home [10], Entropia [11], Farsite [12], BOINC [13], CCOF [14], Kosha [15], OurGrid [16], Alchemi [17],FreeLoader [18], LHC@home [19], Aneka [20], Cohesion [21], EDGeS [22], BitDew [23], unaGrid [24], ATLAS@home [25]
Volunteer cloud computing	Storage@home [26], Cloud@home [27], Seattle [28], C3 [29], P3R3.O.KOM [30], STACEE [31], UnaCloud [32], Personal Cloud [33], P2PCS [34], SoCVC [35], Fatman [36], AdHoc Cloud [37], SASCloud [38], DIaaS [39], Nebula [40], cuCloud [41]
Volunteer mobile computing	Mobiscope [42], AnonySense [43], Micro-blog [44], LiveCompare [45], Bubblesensing [46], PRISM [47], CrowdLab [48], CWC [49], Serendipity [50], Mobile Device Clouds [51], CellCloud [52], GEMCloud [53], FemtoCloud [54], AirShower@home [55]
Volunteer parallel computing	VolpexPyMPI [56], MOON [57], BOINC-MR [58], MPIWS [48], ADAPT [59], GiGi-MR [60], freeCycles [61], Adoop [62], CloudFinder [63]

2.2. Reference Model of Volunteer Computing

According to [5,64], VC follows mostly a master–worker parallel computing model as shown in Figure 6a. In this model, the master decomposes massive tasks into small chunks and distributes these small chunks among workers. The master can be itself composed of distributed nodes, but it is a centralized concept for the overall architecture. Workers perform the required computation and send results back. The master then verifies data results and aggregates them to compute final results. The client nodes are inherently claimed by the master and used exclusively for a specific VC project. By principle, it would be possible to install different VC client software on clients to contribute to various VC projects. However, this is conceptually not considered and might end in problems. For example, it is not clear whether and how different VC projects would be prioritized.

Middleware (see Figure 6b) handles all operations of a VC system. These operations compromise splitting tasks into small chunks, scheduling these chunks and aggregating the chunk results, and protecting privacy and security of VC computing devices [64]. However, it is also essential to process truthfully. Thus, claimed applications and claimed work done by the middleware of a project should be transparently stated. This truthfulness includes addressing questions like possible

economic exploitation, expected results, the surrender of intellectual property, and similar aspects [64]. For example, a VC project should not claim to research about cancer but do crypto-mining instead.



(a) Exclusive use of volunteer workers **(b)** Master worker architecture

Figure 6. Common master worker model used by many VC projects.

Most VC projects rely on a controllable and exclusive environment to achieve this trust technically. They provide the master nodes, and worker nodes attach to these master nodes exclusively. The trust is expressed by installing the VC client software necessary to act as a worker for a specific VC project. Consequently, the worker is bound to one particular project, and it is not handed over to another VC project even if the overall workload would suggest this. Even if different VC projects share the same middleware, this work is not frictionless. Consequently, VC projects that share no common middleware are isolated (see Figure 6a).

2.3. Open Problems in Volunteer Computing

The COVID-19 case (Folding@home), recent reviews [5,64], and the critical discussion of the current state of VC turned up the following (not necessarily complete) shortcomings of current VC platforms.

- **Heterogeneity [64]:** Different VC devices have different power, memory and processing capabilities, as well as different communication interfaces, making it hard to classify, design, and assign device optimized work units.
- **Result verification [64]:** Volunteers perform their required computation and send data results back to the master. The master then verifies data results and discards inadequate or erroneous results. In this way, massive computation (several hours or even days are not unlikely) is wasted as result verification is done at the end of processing. Intermediate result verification mechanisms or smaller chunks could minimize this waste.
- **Project exclusiveness:** Currently, for most VC platforms, projects are organizations (usually academic research groups) that need computational power. Each project runs on project-dedicated master servers. On the one hand, this enables trust, but hinders sharing of devices across different projects. This exclusiveness can result in situations that the Folding@home project faced in March and April 2020 due to the massive COVID-19 scale-out. In this situation, all of the unused CPU

cycles (see Figure 2) could have been provided with ease to other VC projects dedicated to further respectable motives. Thus, in March and April 2020, plenty of possible computations for cancer, climate change, and further research projects could have been processed without disadvantaging COVID-19 research. The COVID-19 pipelines were almost empty in that phase because there were more devices than tasks. There was nothing to process.

- **Security:** Good security is multi-layered. For instance, the BOINC system maintains reasonable security practices at several levels (https://boinc.berkeley.edu/trac/wiki/SecurityIssues). Let us investigate them from a best-practice point of view.

  All VC systems require that donators have to run executables provided by a third party—the company or institution running the project. Thus, these third party executables are highly suspicious from a security point of view. The following counter-measures are combined to mitigate corresponding risks.

  - Overall **project security** measures very often include security audits of project code, enforcing SSL communication with project infrastructure, and virus scanning of project files.
  - **Code signing** can be used to provide valid official builds and to detect code injection attacks on the client-side.
  - **Result verification** is used on the master side to verify that malicious clients have not manipulated results.
  - **Sandboxing** can limit the risk for donators from malicious or insecure project code. However, sandboxing must be very client operation system-specific.

  However, most VC projects are operated by domain-matter experts and not by IT-security experts. It would be a benefit for the VC projects (reduced efforts) and the donators (improved security) if both sides could rely on proven security infrastructures.
- **Scalability and elasticity [5]:** VC systems can have millions of volunteer nodes connected to them. Moreover, this amount of nodes can grow exponentially and quickly as COVID-19 taught us. Thus, more scalable and elastic approaches are required to handle this significant number of volunteer nodes coupled with their intermittent availability. Using more decentralized architectures has already been proposed and implemented [13]. However, the Folding@home project still could not scale-out fast enough to handle all of the COVID-19 volunteers (see Figure 2).

3. A Review of the Current State of Cloud Computing

The reader should be aware that this section is mainly a summary of [3] to provide a more convenient reading experience. As it already has been stated, the COVID-19 case of Folding@home disclosed some "lock-in" shortcomings of VC regarding elasticity and efficient use of shared resources. This "lock-in" shows some astonishing parallels with cloud computing. Precisely, these two points (elasticity and resource utilization) have been mainly optimized throughout the last ten years in cloud computing [4]. Theses lessons learned originate in profane economic considerations of cloud-native companies like Netflix, Uber, Airbnb, and many more. However, the resulting technological solutions might be rewarding for VC because they address by accident some of the identified shortcomings. Therefore, we want to focus on these core insights because these insights might be used to address and solve some of the identified problems of VC in Section 2.3.

According to our experiences and action research activities over the last ten years, cloud computing is dominated by two major long-term trends. In the first adoption phase of cloud computing, existing IT-systems were merely transferred to cloud environments. The original design and architecture of these applications were not changed. Applications have only been migrated from dedicated to virtualized hardware. Over the years, cloud system engineers implemented remarkable improvements in cloud platforms (PaaS) and infrastructures (IaaS). In particular, we investigate resource utilization improvements (Section 3.1) and the architectural evolution of cloud applications (Section 3.2).

3.1. A Review of the Resource Utilization Evolution

Cloud-native applications are built to be elastic. If not, cloud computing would very often not be reasonable from an economic point of view [65]. Elasticity is understood as the degree to which a system adapts to workload changes. Over time, systems were designed intentionally for such elastic cloud infrastructures. Accordingly, the utilization rates of underlying computing infrastructures increased. New deployment and design approaches like containers, microservices, or serverless architectures evolved.

Figure 7 shows a noticeable trend over the last decade. Machine virtualization consolidated plenty of bare-metal machines and formed the technological backbone of IaaS cloud computing. Virtual machines might be more lightweight than bare metal servers. However, containers are much more fine-grained and improved two things: The way of standardized deployments, but they also increased the utilization rates of virtual machines. Nevertheless, containers are still always-on components. Thus, Function-as-a-Service (FaaS) approaches evolved and introduced time-sharing concepts in container platforms. Using FaaS, units are only executed if requests have to be processed. Therefore, FaaS enables scale-to-zero deployments and improves resource efficiency [66]. Thus, the technology stack—although getting more complicated—followed the trend to run more workload on the same amount of physical machines by shrinking the size of standardized deployment units, be it virtual machines, containers, or functions.

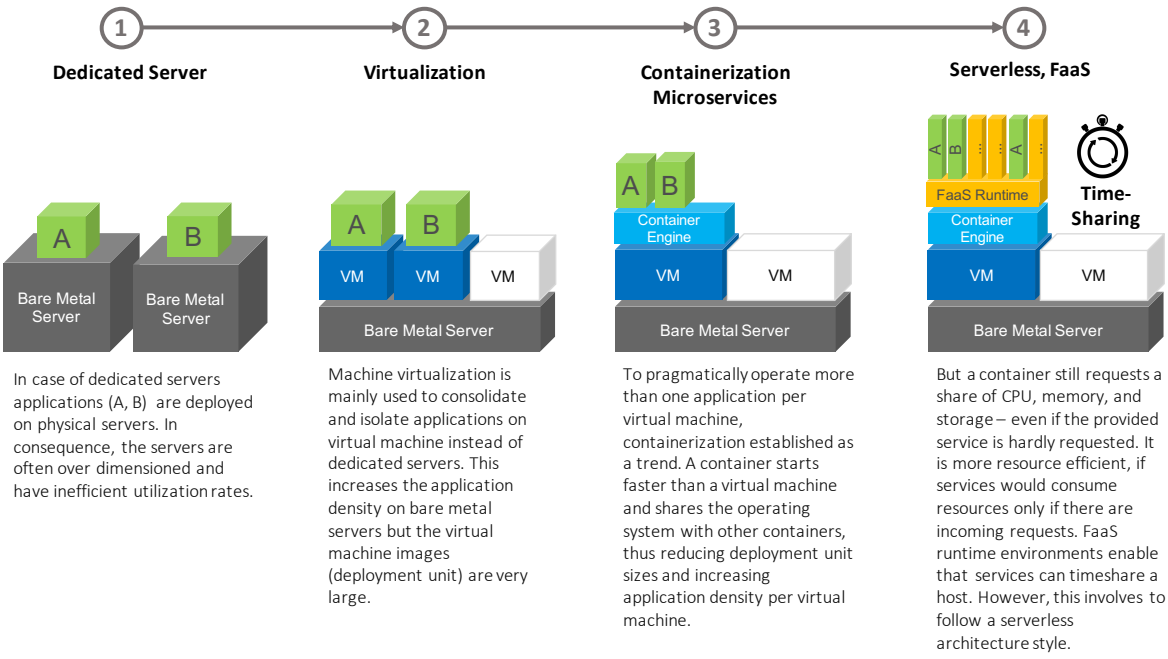


Figure 7. Observable trend of minimizing deployment unit sizes, taken from [3].

This **virtual-machine** → **container** → **function** resource utilization evolution is accompanied by corresponding architectural approaches:

- **Service-oriented architectures (SOA)** fitted very well with monolithic deployment approaches that can be provided using standardized virtual machines (IaaS).
- **Microservice architectures** are built on top of loosely coupled and independently deployable services. These services can be provided via much smaller and standardized containers. We could rate Microservices as a kind of standardized PaaS cloud service provision model.
- Finally, **serverless architectures** are mainly event-driven service-of-service architectures where their functionality is provided as "nano"-services via functions. Serverless and FaaS are the latest trends in cloud computing, so functions are not standardized yet. However, more and more

Cloud-native computing foundation (CNCF) hosted serverless approaches like Kubeless (https://kubeless.io), Knative (https://knative.dev), or OpenWhisk (https://openwhisk.apache.org) make use of containers to package and deploy functions. Thus, it seems likely that containers might evolve as the de-facto deployment unit format not only for microservices, but also for functions.

3.2. A Review of the Architectural Evolution

The reader observes that cloud-native applications aim for better resource utilization by applying more fine-grained deployment units—for instance, containers instead of virtual machines or functions instead of containers. Improvements in resource utilization rates always had an impact on cloud-native architecture styles. Let us now investigate the two major architectural trends that might be of most interest from a VC perspective.

3.2.1. Microservice Architectures

Microservices form *"an approach to software and systems architecture that builds on the well-established concept of modularization but emphasizes technical boundaries. Each module—each microservice—is implemented and operated as a small yet independent system, offering access to its internal logic and data through a well-defined network interface. This architectural style increases software agility because each microservice becomes an independent unit of development, deployment, operations, versioning, and scaling [67]."* Faster delivery, improved scalability, and greater autonomy are often mentioned benefits of microservice architectures [67,68]. Different services are independently scalable based on actual request stimuli. Because services can be developed and operated by different teams, they not only have a technological but also an organizational impact. Thus, localized decisions per service regarding programming languages, libraries, frameworks, and more are possible and enable best-of-breed approaches.

Besides the pure architectural point of view, the following tools, frameworks, services, and platforms form the current understanding of the term microservice:

- Service discovery technologies decouple services from each other. Services must not explicitly refer to network locations.
- Container orchestration technologies automate container allocation and management tasks.
- Monitoring technologies enable runtime monitoring and analysis of the runtime behavior of microservices.
- Latency and fault-tolerant communication libraries enable efficient and reliable service communication in permanently changing configurations.
- Service proxy technologies provide service discovery and fault-tolerant communication features that are exposed over HTTP.

A complex tool-chain evolved to handle the continuous operation of microservice-based cloud applications [3]. We should consider this for VC and took only the barely necessary concepts.

3.2.2. Serverless Architectures

The serverless computing model allocates resources dynamically and intentionally out of control of the service customer. To scale to zero resources might be the most critical differentiator of serverless platforms compared with other IaaS or PaaS-based cloud platforms. This scale-to-zero capability excludes the most expensive always-on usage pattern [65]. Consequently, the term "serverless" is getting more and more attraction [67]. However, where have all the servers gone? Processing resources must still exist somehow.

Serverless architectures make substantial use of Function-as-a-Service (FaaS) concepts and platforms [69] and integrate more intensively third-party backend services. Figure 7 shows this evolution over the last ten years. FaaS platforms realize time-sharing of resources and increase the utilization factor of computing infrastructures. Cost reductions of 70% are possible [66].

A FaaS platform is nothing more than an event processing system (see Figure 8). Serverless platforms take an event and then determine which functions are registered to process the event [70]. If no function is present, a new one is created. Event-based applications are especially very much suited for this approach [70,71].

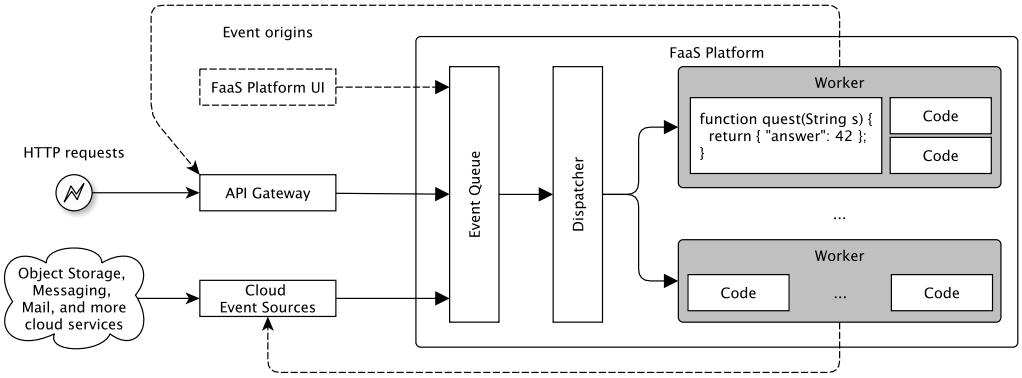


Figure 8. Serverless platform architecture, taken from [3].

In summary, the following observable engineering decisions in serverless architectures are worth being mentioned:

- Cross-sectional logic, like authentication or storage, is sourced to external third party services.
- End-user clients or edge devices do the Service composition. Thus, service orchestration is not done by the service provider but by the service consumer via provided applications.
- Endpoints using HTTP- and REST-based/REST-like communication protocols that can be provided easily via API gateways are generally preferred.
- Only very domain or service-specific functions are provided on FaaS platforms.

Thus, the serverless design is generally more decentralized and distributed. It makes more intentional use of independently provided services and is therefore much more intangible compared with microservice architectures.

In particular, this distribution characteristic seems to make it more suitable for VC. Thus, serverless principles might be more preferable than microservice principles to consider for VC.

4. Discussion of Technological and Architectural Opportunities for Future Volunteer Computing

In Section 3, the reader got to know how the design of so-called cloud-native systems have changed. Technologies that have been massively improved, integrated, and simplified throughout the last ten years are containers, image registries, service registries, and service proxies. The primary motivation for these changes has been economical. If this had not been done, cloud-deployed systems would waste valuable (and costly) cloud resources. Some similar efficiency problems could be observed during COVID-19 crisis in the Folding@home project (see Figure 2). Therefore, this section will investigate whether and how these cloud-native improvements could be used to evolve the VC model that has been summarized in Section 2.2 and Figure 6.

In short, this perspective paper proposes to transform the current situation of isolated VC project networks into a more meshed variant (see Figure 9). Therefore, all VC projects must share their master endpoints in a standardized way. The vision is that a worker should only know one IP address of the global VC network to gain information about all other existing and available VC projects and endpoints. It is then up to the worker to decide to which projects it would like to contribute. In addition, a worker should always be able to contribute to more than one project at a time. Thus, if a favored project does not request resources, the worker could fetch tasks from other VC projects according to a priority list or any different kind of prioritization. If this was possible, the COVID-19 case of the Folding@home

project (see Figure 1) would not have happened. Unused resources had been automatically spent on other projects that address climate change, cryptography, number theory, or whatever.

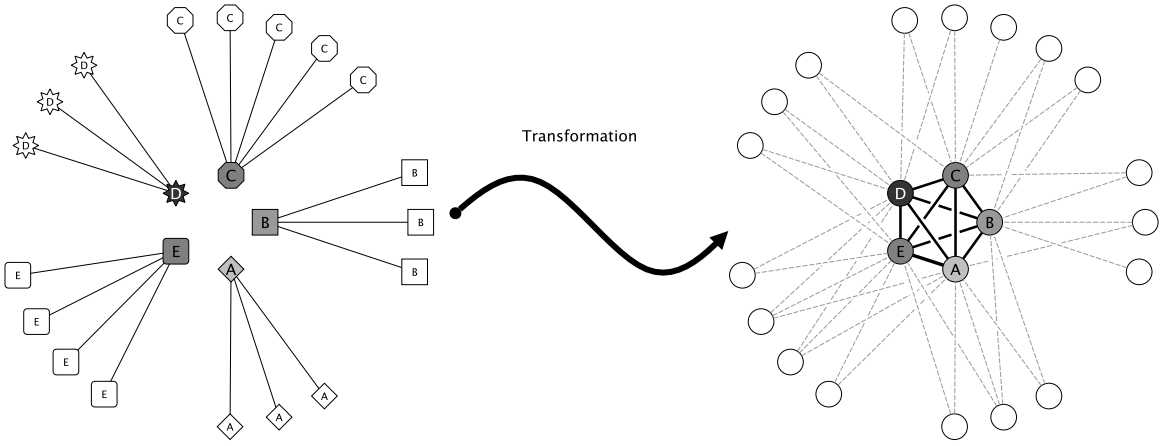


Figure 9. Transformation of VC networks to avoid project boundness of worker nodes.

4.1. Standardization of Deployment Units

Therefore, this perspective paper proposes in Figure 10 an evolved and container-based master–worker architecture extending the typical VC master–worker architecture (see Figure 6).

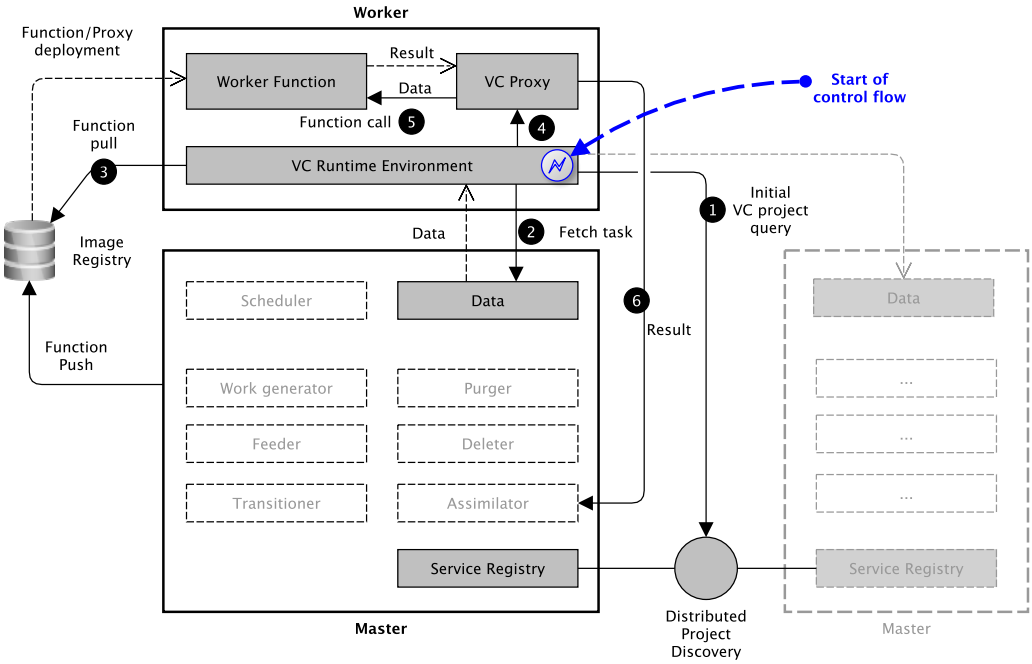


Figure 10. Proposal of an evolved and container-based VC master worker architecture model.

The marked grey components are extending or changing specific parts of the VC reference architecture explained in Section 2.2. First of all, the proposed architecture considers more than one VC project. All master nodes provide a standard Service Registry component that is shared by all VC projects to enable a complete VC project awareness for worker clients. Thus, a worker can choose which project it would like to provide its computing resources. Therefore, all VC projects must provide their workload in a standardized but flexible deployment format to make this freedom of choice possible. Therefore, the architecture proposes to make use of container images as deployment format.

Signed images are provided via public **Image Registries**. A VC project can operate a project-specific image registry. Alternatively, public image registries (like DockerHub https://hub.docker.com) would work as well.

Thus, the worker client can be minimized to a standardized container runtime engine (for example Docker or any other OCI conform container runtime engine) that fetches provided images and executes them as **Worker Functions** in a FaaS-like style (see Figure 8). However, worker functions are accompanied by a so-called side-car container [67] (**VC proxy**) that handles common VC communication patterns and can be called by the **VC Runtime Environment** (a lightweight wrapper around an OCI conform container runtime environment). VC proxy and the VC Runtime Environment are designed to be standardized components that must be not adapted by a VC project. The VC proxy provides a standard interface furthermore for worker functions. Obviously, the worker function is a VC project-specific part that realizes data processing. However, because the proposal makes use of container technologies, it can be provided as a standardized deployment unit. Container technologies enable furthermore polyglot programming, so there is a freedom of programming language choice for implementing worker functions. Current VC platforms like BOINC often enforce to include specific libraries which are written very often in C/C++.

Table 2 summarizes the selected and discussed cloud-native technologies and explains how these technologies can be used to realize the proposed architecture (see Figure 10) to mitigate mentioned VC open issues reported in the literature. To do this, well-tried technologies that form the backbone of modern cloud-native architectures could be adopted: Containers, even more fine-grained (but container-based) functions, image registries, and distributed service registries seem very promising here. Thus, like the BOINC-approach, a middleware-based approach is proposed to share resources between different VC projects. However, most parts of this middleware are already existing (the technologies mentioned above and listed in Table 2). Thus, the proposal does not require a complete new middleware or framework. Still, the proposal requires a VC-specific integration of these technologies that go beyond the BOINC-approach (dating back to the early 2000 s, so to a pre-cloud era). This VC-specific integration is called **VC Runtime Environment** in Figure 10.

4.2. Client-Side Service Discovery Initiated Workflow

A publish–subscribe communication model between the client and the master nodes is nearby. However, the publish–subscriber model assumes to some degree that both client- and master-components are (not perfectly but to some degree reliable) always-on components. This assumption is not entirely the case in volunteer computing, especially not on the client-side. Thus, the proposed approach follows the publish–subscriber philosophy but evolves it as a straightforward and purely client-side triggered approach. In VC, the clients form the ephemeral parts of the complete system. Taking the insights of [72], one can expect simply less error tracking efforts if the ephemeral components (clients) query and request the stable parts (masters).

In VC, we are talking about hundreds of VC projects operating thousands of master nodes that mainly operate in an always-on mode. On the worker side, we are talking about a much more volatile setting of millions of client nodes that are only sporadically available. Therefore, we generally advocate client-side service discovery and distributed server-side service registries because the unchanging parts are more on the master and less on the client-side. Thus, client-side service discovery has to query much less moving and changing roles in this setting (thousands of service endpoints on the master-side instead of millions of service endpoints on the client-side). Whenever a client is available, it can make (well cacheable) client-side service discovery and ask for VC tasks and process them according to the client preferences and priorities. Thus, client-side service discovery can be used to create a naturally occurring workload sharing across different VC projects (and not just within a VC project).

Table 2. Mapping of recent cloud-native technologies to identified VC open issues.

VC Issues	Container	Function	Image Registry	Service Registry	Service Proxy	Remarks
HW heterogenity	x	x				Containers (and functions packaged as containers) are a standardized deployment format that is useable on all primary desktop and server operating platforms (Windows, Linux, Mac OS).
Verification of (large) results		x			x	Functions are used in cloud-native architectures to process events that must be computed in a limited amount of time. Functions (if packaged as containers) can be accompanied by trusted service proxies that could validate function results before sending them to the master. Because of the time limitations (minutes instead of hours or even days), the result verification would be faster and might be even processed decentrally.
Code signing and updating			x			Image content trust technologies provide the ability to use digital signatures for image registry operations (push, pull). Publishers can sign their pushed images, and image consumers can ensure that pulled images are signed. If images are updated they can be fetched automatically by the clients in their next event processing cycle. Current image registries like Harbor, DockerHub, quai.io, GitLab registry, and many more provide signed images for automatic deployments out of the box.
Sandboxing	x	x				The original intent of operating system virtualization (containers) was sandboxing. Thus, containers (and functions packaged as containers) provide inherent and reliable sandboxing out of the box. This sandboxing is much more fine-grained than virtual machines and available on all major desktop and server platforms (see HW heterogeneity).
Project exclusiveness				x		A service registry is a database containing the network locations of service instances. It consists typically of components that use a replication protocol. Examples for reliable cloud-native products are etcd (https://etcd.io), consul (https://www.consul.io), or Zookeeper (https://zookeeper.apache.org). Such solutions can share VC project information and network locations of master nodes for clients in a project agnostic format. Thus, clients that are bound to one master component can do client-side service discovery of further VC projects.

The client-side initiated workflow loops through the steps ❶ to ❻ shown in Figure 10:

1. In Step ❶, the **VC Runtime Environment** of a **worker node** queries periodically (for example, each day, every six hours or similar) the distributed VC project discovery service that is formed by master nodes of various VC projects. This updates a worker node's VC project awareness to decide which master nodes to ask for processing tasks.
2. In Step ❷, the **VC Runtime Environment** of a worker node **selects a master node** according to its updated project awareness and fetches a task (including the data to be processed). If this fails (for instance, the master node might be not available, has no jobs, etc.), **another task from another master node (even from a different project)** is fetched according to worker node preferences.
3. In Step ❸, the **VC Runtime Environment** analysis the task and triggers a corresponding **Function pull** from a **public image registry** to fetch (if not already present) and start the VC project-specific **Worker function** container image. Therefore, the address of the image registry, the unique image name of the **Worker function**, and image version must be part of the task description. Furthermore, the task description must contain the URL of the data to be processed.
4. In Step ❹, the **VC Runtime Environment** handles the control over to a **VC proxy**. This proxy does communication with the Worker function and decouples the runtime environment from the Worker function.
5. In Step ❺, the **VC proxy** calls the **Worker function** with the to be processed **data** and receives the **result**.
6. Finally, in Step ❻, the **VC proxy** can even do the result verification on the Worker-side (and not on the Master). Like the Worker function, the **VC proxy** is simply a container that is instantiated from a trusted image and may, therefore, contain signed result verification logic that cannot be tampered unnoticed. As a last step, the VC proxy transmits the result to the assimilation endpoint of the master node (this endpoint must also be part of the task description).

The process would go on with step ❷ (or step ❶ if a periodic update of the VC project awareness is necessary). It would address the resource sharing problem efficiently and with a simple client-side strategy.

5. Critical Discussion and Related Work

The proposed approach targets mainly volunteer grid computing projects and, in particular, the mentioned shortcomings of cross-project resource sharing. The main intention is to improve resource sharing across VC projects and to set up VC projects more easily making use of established and well-accepted cloud-native technologies. Therefore, it shares similar limitations like every other volunteer grid computing project. Thus, the architecture supports embarrassingly parallel tasks that need no or little communication among the tasks [5]. If this is not the case, volunteer parallel computing projects or even HPC supercomputing might be a better fit. However, this paper does not focus on this kind of parallel VC or even HPC. The paper does not even claim to have or provide answers of value for these parallel computing or HPC supercomputing domains.

Furthermore, the proposed approach follows conceptually a middleware-based path and shares, therefore, comparable limitations to the BOINC-approach [13,73]. To reach project awareness and enable necessary trust are complex tasks in themselves, even in a single-research VC project. Single-research projects might even have advantages here because contributors do not need any complex overview of various research or other projects. However, this aspect is getting more complicated for a middleware-based approach [74]. The contributors need project awareness, and some project prioritization means. Some contributors want to support disease-related projects but might be not interested in supporting prime-number mathematical research. In the case of BOINC, a kind of trusted community already exists, and all BOINC-based projects have passed a kind of quality gate, and all BOINC-based projects are research-focused. This gatekeeper role of BOINC makes it easier for BOINC contributors to establish trust. However, projects that are not aware of the gatekeeper are not aware to contributors as well. Thus, multi-project gatekeeping in VC always has a kind of censorship.

Therefore, the proposed approach enables intentionally to set up VC infrastructures that could process arbitrary computations—for instance, doing mundane crypto-mining for purely monetary reasons. We do not think that crypto-mining (or other doubtable motivations) would be a useful and ethical form of VC. However, this should not be the decision of a non-democratic legitimized gatekeeping institution but the personal choice of every single VC contributor according to their own criteria [75,76].

Therefore, this proposal intentionally does not assume a well-trusted gatekeeper that filters qualified from non-qualified projects. However, this missing gatekeeping role makes it more complicated for contributors to select VC projects that are worth being supported.

The reader should take additional surveys on VC, like [5,64] or cloud computing [2,3,67,69,70], into account to derive their own conclusions and discuss this perspective paper critically. For instance, Ref. [5] provides a broad and excellent overview of several grid-based, cloud-based, mobile, and parallel VC projects, frameworks, and technological approaches. As the reader may have noticed, this perspective paper is highly influenced by [5]. However, to the best of the author's knowledge, there does not exist any survey that covers cloud computing and VC in parallel focusing particularly on the aspect of how to combine concepts of both domains to overcome the project exclusiveness problem in VC.

It is interesting to see that grid- and cloud-based projects form the majority of VC projects (see Figure 4). Thus, there is some kind of attraction in adopting Cloud technologies for VC (see Figure 5). In particular, recent cloud-native trends like standardization of fine-grained deployment units via containers provide exciting opportunities. *"The efficient use of available resources and tight integration with the host operating system makes container technologies a plausible choice for VC. [...] More research is important to investigate the suitability and efficiency of container technologies for VC, specifically for volunteer cloud systems [64]".*

However, this paper shifted the focus less on volunteer clouds but postulated to adopt container technologies in VC to improve the overall share- and portability between VC projects to enable overflow processing between different projects. Projects like [77] strive to do something similar by integrating supercomputing with VC and Cloud Computing. However, their focus is more on how to make the high-performance end of supercomputing data centers available for VC.

6. Conclusions

The COVID-19 pandemic created the largest volunteer supercomputer on earth. However, this largest supercomputer on the planet ran idle for a significant amount of time—what a waste of resources. Therefore, this perspective paper investigated how the sharing of donated resources across VC projects could be improved. If one cannot fight COVID-19, one might contribute to mathematical research, climate research, other disease research, or any different computationally intensive kind of research. Most of the VC donators will provide their resources regardless of the specific aim of a VC project.

This perspective paper proposes to tackle the disclosed resource sharing shortcomings of volunteer computing using technologies that have been invented, optimized, and adapted for entirely different purposes by cloud-native companies like Uber, Airbnb, Google, or Facebook. Such promising technologies might be containers, serverless architectures, image registries, distributed service registries that can address problems like hardware heterogeneity, sandboxing, code signing and updating, result verification, and most importantly to overcome project exclusiveness. All these cloud-native technologies mentioned have one thing in common: They already exist and are all tried and tested in large web-scale deployments.

However, the reader should keep in mind that this paper is a perspective paper. It does not present a validated solution proposal intentionally. Nevertheless, it offered a detailed list of technologies to overcome current shortcomings of VC that the COVID-19 case disclosed. This concrete strategy does not claim the "philosopher's stone" but strives to foster discussions in the VC community on how

VC could transform into a more global VC grid of cooperating VC projects that share and do not claim resources. In addition, a lot of interesting research questions will appear if this path is followed. This path compromises the fact that each VC project is still operating their own master nodes and control plane infrastructure but yielding access to donated resources.

Consequently, a win–win-situation can be expected: Future VC projects should gain access to a much broader set of donated resources. COVID-19 showed us that VC projects can easily build the biggest supercomputer on earth. In addition, VC donors would gain access to a much more multifarious spectrum of research projects. The middleware must simply be more standardized, and more focused on resource sharing. Cloud computing has followed this exact path successfully for a bit more than a decade. Perhaps VC should also have a look?

Funding: This research received no external funding.

Acknowledgments: Let me thank our students at the Lübeck University of Applied Sciences to support the Folding@home project and Sars-CoV-2 research providing precious assets: Gaming PCs with powerful GPUs. However, the primordial and likely unconscious initiator of this paper was Josef Adersberger (we normally cooperate in several cloud-native publication projects). He is the CEO of the cloud-native consulting company QAware GmbH and a cloud-native computing expert. During the COVID-19 pandemic, employees of QAware formed a Folding@home group to support Sars-CoV-2 research, and Josef Adersberger posted their company rank (top 3% of worldwide contributors) on Twitter. We accepted his "challenge" and supported the Folding@home project as well (like many others worldwide). Without this "challenge", this perspective paper would never have been written.

Conflicts of Interest: The author declares no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

API	Application Programming Interface
BOINC	Berkeley Open Infrastructure for Network Computing (https://boinc.berkeley.edu)
CNCF	Cloud-Native Computing Foundation (https://cncf.io)
HTTP	Hypertext Transfer Protocol
HW	Hardware
IaaS	Infrastructure as a Service
IP	Internet Protocol
OCI	Open Container Initiative (https://opencontainers.org)
PaaS	Platform as a Service
REST	Representational State Transfer
SaaS	Software as a Service
SOA	Service Oriented Architecture
URL	Uniform Resource Locator
VC	Volunteer Computing
QoS	Quality of Service

References

1. Kratzke, N.; Siegfried, R. Towards cloud-native simulations–lessons learned from the front-line of cloud computing. *J. Def. Model. Simul.* **2020**. [CrossRef]
2. Kratzke, N.; Quint, P.C. Understanding Cloud-native Applications after 10 Years of Cloud Computing—A Systematic Mapping Study. *J. Syst. Softw.* **2017**, *126*, 1–16. [CrossRef]
3. Kratzke, N. A Brief History of Cloud Application Architectures. *Appl. Sci.* **2018**, *8*, 1368. [CrossRef]
4. Kratzke, N.; Quint, P.C. *Technical Report of Project CloudTRANSIT-Transfer Cloud-Native Applications at Runtime*; Technical Report; Lübeck University of Applied Sciences: Lübeck, Germany, 2018. [CrossRef]
5. Mengistu, T.M.; Che, D. Survey and Taxonomy of Volunteer Computing. *ACM Comput. Surv.* **2019**, *52*. [CrossRef]
6. Project, B. Publications by BOINC Projects. 2020. Available online: https://boinc.berkeley.edu/wiki/Project_list (accessed on 5 June 2020).

7. Chu, D.C.; Humphrey, M. Mobile OGSI.NET: Grid computing on mobile devices. In Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA, USA, 8 November 2004; pp. 182–191.
8. Litzkow, M.J.; Livny, M.; Mutka, M.W. Condor-a hunter of idle workstations. In Proceedings of the 8th International Conference on Distributed, San Jose, CA, USA, 13–17 June 1988; pp. 104–111.
9. Fedak, G.; Germain, C.; Neri, V.; Cappello, F. XtremWeb: A generic global computing system. In Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid, Brisbane, Australia, 15–18 May 2001; pp. 582–587.
10. Anderson, D.P.; Cobb, J.; Korpela, E.; Lebofsky, M.; Werthimer, D. SETI@home: An Experiment in Public-Resource Computing. *Commun. ACM* **2002**, *45*, 56–61. [CrossRef]
11. Chien, A.; Calder, B.; Elbert, S.; Bhatia, K. Entropia: Architecture and performance of an enterprise desktop grid system. *J. Parallel Distrib. Comput.* **2003**, *63*, 597–610. [CrossRef]
12. Adya, A.; Bolosky, W.J.; Castro, M.; Cermak, G.; Chaiken, R.; Douceur, J.R.; Howell, J.; Lorch, J.R.; Theimer, M.; Wattenhofer, R.P. Farsite: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. *SIGOPS Oper. Syst. Rev.* **2003**, *36*, 1–14. [CrossRef]
13. Anderson, D.P. BOINC: A system for public-resource computing and storage. In Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA, USA, 8 November 2004; pp. 4–10.
14. Zhou, D.; Lo, V. Cluster Computing on the Fly: Resource discovery in a cycle sharing peer-to-peer system. In Proceedings of the IEEE International Symposium on Cluster Computing and the Grid, Chicago, IL, USA, 19–22 April 2004; pp. 66–73.
15. Butt, A.R.; Johnson, T.A.; Zheng, Y.; Hu, Y.C. Kosha: A Peer-to-Peer Enhancement for the Network File System. In Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, Pittsburgh, PA, USA, 6–12 November 2004; p. 51.
16. Andrade, N.; Costa, L.; Germoglio, G.; Cirne, W. Peer-to-peer grid computing with the ourgrid community. In Proceedings of the SBRC 2005-IV Salao de Ferramentas, Agia Napa, Cyprus, 31 October–4 November 2005.
17. Luther, A.; Buyya, R.; Ranjan, R.; Venugopal, S. Alchemi: A. NET-based Enterprise Grid Computing System. In Proceedings of the 6th International Conference on Internet Computing (ICOMP'05), Las Vegas, NV, USA, 27–30 June 2005.
18. Vazhkudai, S.S.; Ma, X.; Freeh, V.W.; Strickland, J.W.; Tammineedi, N.; Scott, S.L. FreeLoader: Scavenging Desktop Storage Resources for Scientific Data. In Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, Seattle, WA, USA, 12–18 November 2005; p. 56.
19. Herr, W.; McIntosh, E.; Schmidt, F.; Kaltchev, D. Large Scale Beam-Beam Simulations for the Cern LHC Using DIstributed Computing. In Proceedings of the 10th European Particle Accelerator Conference, Edinburgh, UK, 26–30 June 2006.
20. Chu, X.; Nadiminti, K.; Jin, C.; Venugopal, S.; Buyya, R. Aneka: Next-Generation Enterprise Grid Platform for e-Science and e-Business Applications. In Proceedings of the Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007), Washington, DC, USA, 10–13 December 2007; pp. 151–159.
21. Schulz, S.; Blochinger, W.; Held, M.; Dangelmayr, C. COHESION—A microkernel based Desktop Grid platform for irregular task-parallel applications. *Future Gener. Comput. Syst.* **2008**, *24*, 354–370. [CrossRef]
22. Urbah, E.; Kacsuk, P.; Farkas, Z.; Fedak, G.; Kecskemeti, G.; Lodygensky, O.; Marosi, C.A.; Balaton, Z.; Caillat, G.; Gombás, G.; et al. EDGeS: Bridging EGEE to BOINC and XtremWeb. *J. Grid Comput.* **2009**, *7*, 335–354. [CrossRef]
23. Fedak, G.; He, H.; Cappello, F. BitDew: A Data Management and Distribution Service with Multi-Protocol File Transfer and Metadata Abstraction. *J. Netw. Comput. Appl.* **2009**, *32*, 961–975. [CrossRef]
24. Castro, H.; Rosales, E.; Villamizar, M.; Jiménez, A. UnaGrid: On Demand Opportunistic Desktop Grid. In Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, Melbourne, Australia, 2–5 November 2010; pp. 661–666.
25. Adam-Bourdarios, C.; Cameron, D.; Filipčič, A.; Lancon, E.; Wu, W. ATLAS@Home: Harnessing Volunteer Computing for HEP. *J. Phys. Conf. Ser.* **2015**, *664*, 022009. [CrossRef]
26. Beberg, A.L.; Pande, V.S. Storage@home: Petascale Distributed Storage. In Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium, Hagenberg, Austria, 5–8 June 2007; pp. 1–6.

27. Cunsolo, V.D.; Distefano, S.; Puliafito, A.; Scarpa, M. Volunteer Computing and Desktop Cloud: The Cloud@Home Paradigm. In Proceedings of the 2009 Eighth IEEE International Symposium on Network Computing and Applications, Cambridge, MA, USA, 9–11 July 2009; pp. 134–139.
28. Cappos, J.; Beschastnikh, I.; Krishnamurthy, A.; Anderson, T. Seattle: A Platform for Educational Cloud Computing. *SIGCSE Bull.* **2009**, *41*, 111–115. [CrossRef]
29. Briscoe, G.; Marinos, A. Digital ecosystems in the clouds: Towards community cloud computing. In Proceedings of the 2009 3rd IEEE International Conference on Digital Ecosystems and Technologies, Istanbul, France, 1–3 June 2009; pp. 103–108.
30. Graffi, K.; Stingl, D.; Gross, C.; Nguyen, H.; Kovacevic, A.; Steinmetz, R. Towards a P2P Cloud: Reliable Resource Reservations in Unreliable P2P Systems. In Proceedings of the 2010 IEEE 16th International Conference on Parallel and Distributed Systems, Shanghai, China, 8–10 December 2010; pp. 27–34.
31. Neumann, D.; Bodenstein, C.; Rana, O.F.; Krishnaswamy, R. STACEE: Enhancing Storage Clouds Using Edge Devices. In Proceedings of the 1st ACM/IEEE Workshop on Autonomic Computing in Economics, Karlsruhe, Germany, 14–18 June 2011; Association for Computing Machinery: New York, NY, USA, 2011; pp. 19–26. [CrossRef]
32. Osorio, J.D.; Castro, H.; Brasileiro, F. Perspectives of UnaCloud: An Opportunistic Cloud Computing Solution for Facilitating Research. In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), Ottawa, ON, Canada, 13–16 May 2012; pp. 717–718.
33. Hari, A.; Viswanathan, R.; Lakshman, T.V.; Chang, Y.J. The Personal Cloud: Design, Architecture and Matchmaking Algorithms for Resource Management. In Proceedings of the 2nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, San Jose, CA, USA, 24 April 2012; USENIX Association: Berkeley, CA, USA, 2012; p. 3.
34. Babaoglu, O.; Marzolla, M.; Tamburini, M. Design and Implementation of a P2P Cloud System. In Proceedings of the 27th Annual ACM Symposium on Applied Computing, Riva del Garda (Trento), Italy, 26–30 March 2012; Association for Computing Machinery: New York, NY, USA, 2012; pp. 412–417. [CrossRef]
35. Chard, R.; Bubendorfer, K.; Chard, K. Experiences in the design and implementation of a Social Cloud for Volunteer Computing. In Proceedings of the 2012 IEEE 8th International Conference on E-Science, Chicago, IL, USA, 8–12 October 2012; pp. 1–8.
36. Qin, A.; Hu, D.; Liu, J.; Yang, W.; Tan, D. Fatman: Building Reliable Archival Storage Based on Low-Cost Volunteer Resources. *J. Comput. Sci. Technol.* **2015**, *30*, 273–282. [CrossRef]
37. McGilvary, G.A.; Barker, A.; Atkinson, M. Ad Hoc Cloud Computing. In Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing, New York, NY, USA, 27 June–2 July 2015; pp. 1063–1068.
38. Al Noor, S.; Hossain, M.M.; Hasan, R. SASCloud: Ad Hoc Cloud as Secure Storage. In Proceedings of the 2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom), Atlanta, GA, USA, 8–10 October 2016; pp. 37–44.
39. Kim, H.W.; Han, J.; Park, J.H.; Jeong, Y.S. DIaaS: Resource Management System for the Intra-Cloud with On-Premise Desktops. *Symmetry* **2017**, *9*, 8. [CrossRef]
40. Jonathan, A.; Ryden, M.; Oh, K.; Chandra, A.; Weissman, J. Nebula: Distributed Edge Cloud for Data Intensive Computing. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *28*, 3229–3242. [CrossRef]
41. Mengistu, T.; Alahmadi, A.; Alsenani, Y.; Albuali, A.; Che, D. cuCloud: Volunteer Computing as a Service (VCaaS) System. In *International Conference on Cloud Computing*; Springer: Cham, Switzerland, 2018.
42. Agapie, E.; Chen, G.; Houston, D.; Howard, E.; Kim, J.H.; Mun, M.Y.; Mondschein, A.; Reddy, S.; Rosario, R.; Ryder, J.; et al. Seeing our signals: Combining location traces and web-based models for personal discovery. In Proceedings of the 9th workshop on Mobile Computing Systems and Applications, Napa Valley, CA, USA, 25–26 February 2008.
43. Cornelius, C.; Kapadia, A.; Kotz, D.; Peebles, D.; Shin, M.; Triandopoulos, N. Anonysense: Privacy-Aware People-Centric Sensing. In *MobiSys '08, Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services, Breckenridge, CO, USA, 17–20 June 2008*; Association for Computing Machinery: New York, NY, USA, 2008; pp. 211–224. [CrossRef]
44. Gaonkar, S.; Li, J.; Choudhury, R.R.; Cox, L.; Schmidt, A. Micro-Blog: Sharing and Querying Content Through Mobile Phones and Social Participation. In Proceedings of the ACM 6th International Conference on Mobile Systems, Applications, and Services (MOBISYS '08), Breckenridge, CO, USA, 17–20 June 2008.

45. Deng, L.; Cox, L.P. LiveCompare: Grocery Bargain Hunting through Participatory Sensing. In *HotMobile '09, Proceedings of the 10th Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, USA, 23 February 2009*; Association for Computing Machinery: New York, NY, USA, 2009. [CrossRef]
46. Lu, H.; Lane, N.D.; Eisenman, S.B.; Campbell, A.T. Fast Track Article: Bubble-Sensing: Binding Sensing Tasks to the Physical World. *Pervasive Mob. Comput.* **2010**, *6*, 58–71. [CrossRef]
47. Das, T.; Mohan, P.; Padmanabhan, V.N.; Ramjee, R.; Sharma, A. PRISM: Platform for remote sensing using smartphones. In Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, San Francisco, CA, USA, 15–18 June 2010.
48. Calderón, A.; García-Carballeira, F.; Bergua, B.; Sánchez, L.M.; Carretero, J. Expanding the Volunteer Computing Scenario: A Novel Approach to Use Parallel Applications on Volunteer Computing. *Future Gener. Comput. Syst.* **2012**, *28*, 881–889. [CrossRef]
49. Arslan, M.Y.; Singh, I.; Singh, S.; Madhyastha, H.V.; Sundaresan, K.; Krishnamurthy, S.V. Computing While Charging: Building a Distributed Computing Infrastructure Using Smartphones. In Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, Nice, France, 10–13 December 2012; Association for Computing Machinery: New York, NY, USA, 2012; pp. 193–204. [CrossRef]
50. Shi, C.; Lakafosis, V.; Ammar, M.H.; Zegura, E.W. Serendipity: Enabling Remote Computing among Intermittently Connected Mobile Devices. In Proceedings of the Thirteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing, Hilton Head, SC, USA, 11–14 June 2012; Association for Computing Machinery: New York, NY, USA, 2012; pp. 145–154. [CrossRef]
51. Mtibaa, A.; Fahim, A.; Harras, K.A.; Ammar, M.H. Towards Resource Sharing in Mobile Device Clouds: Power Balancing across Mobile Devices. *SIGCOMM Comput. Commun. Rev.* **2013**, *43*, 51–56. [CrossRef]
52. Noor, S.A.; Hasan, R.; Haque, M.M. CellCloud: A Novel Cost Effective Formation of Mobile Cloud Based on Bidding Incentives. In Proceedings of the 2014 IEEE 7th International Conference on Cloud Computing, Anchorage, AK, USA, 27 June–2 July 2014; pp. 200–207.
53. Funai, C.; Tapparello, C.; Ba, H.; Karaoglu, B.; Heinzelman, W. Extending volunteer computing through mobile ad hoc networking. In Proceedings of the 2014 IEEE Global Communications Conference, Austin, TX, USA, 8–12 December 2014; pp. 32–38.
54. Habak, K.; Ammar, M.; Harras, K.A.; Zegura, E. Femto Clouds: Leveraging Mobile Devices to Provide Cloud Service at the Edge. In Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing, New York, NY, USA, 27 June–2 July 2015; pp. 9–16.
55. Gordienko, N.; Lodygensky, O.; Fedak, G.; Gordienko, Y. Synergy of volunteer measurements and volunteer computing for effective data collecting, processing, simulating and analyzing on a worldwide scale. In Proceedings of the 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 25–29 May 2015; pp. 193–198.
56. LeBlanc, T.P.; Subhlok, J.; Gabriel, E. A High-Level Interpreted MPI Library for Parallel Computing in Volunteer Environments. In Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, Melbourne, VIC, Australia, 17–20 May 2010; pp. 673–678.
57. Lin, H.; Ma, X.; Archuleta, J.S.; chun Feng, W.; Gardner, M.K.; Zhang, Z. MOON: MapReduce on Opportunistic eNvironments. In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, Chicago, IL, USA, 21–25 June 2010.
58. Costa, F.; Silva, L.; Dahlin, M. Volunteer Cloud Computing: MapReduce over the Internet. In Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, Shanghai, China, 16–20 May 2011; pp. 1855–1862.
59. Jin, H.; Yang, X.; Sun, X.; Raicu, I. ADAPT: Availability-Aware MapReduce Data Placement for Non-dedicated Distributed Computing. In Proceedings of the 2012 IEEE 32nd International Conference on Distributed Computing Systems, Macau, China, 18–21 June 2012; pp. 516–525.
60. Costa, F.; Veiga, L.; Ferreira, P. Internet-scale support for map-reduce processing. *J. Internet Serv. Appl.* **2013**, *4*, 18. [CrossRef]
61. Bruno, R.; Ferreira, P. FreeCycles: Efficient Data Distribution for Volunteer Computing. In Proceedings of the Fourth International Workshop on Cloud Data and Platforms, Amsterdam, The Netherlands, 13–16 April 2014; Association for Computing Machinery: New York, NY, USA, 2014. [CrossRef]

62. Hamdaqa, M.; Sabri, M.M.; Singh, A.; Tahvildari, L. Adoop: MapReduce for Ad-Hoc Cloud Computing. In Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering, Markham, ON, Canada, 2–4 November 2015; IBM Corp.: New York, NY, USA, 2015; pp. 26–34.
63. Rezgui, A.; Davis, N.; Malik, Z.; Medjahed, B.; Soliman, H. CloudFinder: A System for Processing Big Data Workloads on Volunteered Federated Clouds. *IEEE Trans. Big Data* **2017**, *6*, 347–358. [CrossRef]
64. Nouman Durrani, M.; Shamsi, J.A. Review: Volunteer Computing: Requirements, Challenges, and Solutions. *J. Netw. Comput. Appl.* **2014**, *39*, 369–380. [CrossRef]
65. Weinman, J. Mathematical Proof of the Inevitability of Cloud Computing. Available online: http://www.joeweinman.com/Resources/Joe_Weinman_Inevitability_Of_Cloud.pdf (accessed on 11 May 2020).
66. Villamizar, M.; Garcés, O.; Ochoa, L.; Castro, H.; Salamanca, L.; Verano, M.; Casallas, R.; Gil, S.; Valencia, C.; Zambrano, A.; et al. Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures. In Proceedings of the 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, Colombia, 16–19 May 2016; pp. 179–182.
67. Jamshidi, P.; Pahl, C.; Mendonça, N.C.; Lewis, J.; Tilkov, S. Microservices: The Journey So Far and Challenges Ahead. *IEEE Softw.* **2018**, *35*, 24–35. [CrossRef]
68. Taibi, D.; Lenarduzzi, V.; Pahl, C. *Architectural Patterns for Microservices: A Systematic Mapping Study*; SCITEPRESS: Setúbal, Portugal, 2018.
69. Roberts, M.; Chapin, J. *What Is Serverless?* O'Reilly Media, Incorporated: Sebastopol, CA, USA, 2017.
70. Baldini, I.; Castro, P.; Chang, K.; Cheng, P.; Fink, S.; Ishakian, V.; Mitchell, N.; Muthusamy, V.; Rabbah, R.; Slominski, A.; et al. Serverless computing: Current trends and open problems. In *Research Advances in Cloud Computing*; Springer: Berlin, Germany, 2017; pp. 1–20.
71. Baldini, I.; Castro, P.; Cheng, P.; Fink, S.; Ishakian, V.; Mitchell, N.; Muthusamy, V.; Rabbah, R.; Suter, P. Cloud-native, event-based programming for mobile applications. In Proceedings of the International Conference on Mobile Software Engineering and Systems, Seoul, Korea, 25–26 May 2016; pp. 287–288.
72. Rubab, S.; Hassan, M.F.; Mahmood, A.K.; Shah, S.N.M. A review on resource availability prediction methods in volunteer grid computing. In Proceedings of the 2014 IEEE International Conference on Control System, Computing and Engineering (ICCSCE 2014), Penang, Malaysia, 28–30 November 2014; pp. 478–483.
73. Anderson, D.P. BOINC: A Platform for Volunteer Computing. *J. Grid Comput.* **2019**, *18*, 122–199. [CrossRef]
74. Nov, O.; Anderson, D.; Arazy, O. Volunteer computing: A model of the factors determining contribution to community-based scientific research. In Proceedings of the 19th International Conference on World Wide Web, Raleigh, NC, USA, 26–30 April 2010.
75. Kloetzer, L.; Costa, J.D.; Schneider, D. Not so passive: Engagement and learning in Volunteer Computing projects. *Hum. Comput.* **2016**, *3*, 25–68. [CrossRef]
76. Edinger, J.; Edinger-Schons, L.M.; Schäfer, D.; Stelmaszczyk, A.; Becker, C. Of Money and Morals—The Contingent Effect of Monetary Incentives in Peer-to-Peer Volunteer Computing. In Proceedings of the 52nd Hawaii International Conference on System Sciences, Maui, HI, USA, 8 January–11 Friday 2019.
77. Ritu Arora, C.R. Scalable Software Infrastructure for Integrating Supercomputing with Volunteer Computing and Cloud Computing. *Commun. Comput. Inf. Sci.* **2019**, *964*, 105–119.