Embarrassingly Parallel Grid-computing on the Blockchain

A High-level Overview of the Machinations of XEL

www.xel.org

ABSTRACT

In this overview, we discuss a Blockchain-based, decentralized grid-computing platform which allows tasks with immense computational requirements, usually set in the scientific domain, to utilize a virtually infinite number of computational nodes over large physical distances. In contrast to traditional volunteer-computing where computer owners altruistically donate their spare computational power to one or more research projects, these projects can now create an additional incentive encouraging users to participate by getting rewarded with automated crypto-currency payments in exchange for conducted work. This overcomes the problem that most participants in traditional volunteer-computing network are both typically found in scientific communities and as such very picky about the projects they volunteer to; while an altruistic project like SETI@Home may excite and attract plenty of volunteers, that does not necessarily apply to a "boring" corporate calculation with little to no scientific impact. By introducing crypto-currency payments, our system creates an additional incentive, eliminates a necessary scientific value for attracting volunteers, and contributes to the creation of a system that is open and equally suitable for both scientific and non-scientific use-cases.

Key words. Multiple Data Stream Architectures; Blockchain; Distributed Ledger Technology; Performance

1. Introduction

A vast majority of the world's computational power is not centered around supercomputer centers or laboratories but instead distributed among millions of personal computers in people's homes all around the globe. Volunteer computing[6] is a concept which was first introduced in 1996 by the Great Mersenne Prime Search project [3] and comprises a paradigm which uses these resources to conduct previously infeasible scientific computations. The potential of distributed computation was quickly adopted by other projects such as SETI@home [2], which was launched in 1999 and is dedicated to searching for extra-terrestrial life by analyzing radio signals from different parts of the sky. In 2004, BOINC[1] emerged from the SETI@home project, making this technology accessible to the general public allowing virtually any research project to be aided by distributed computation. Typical for these approaches is that they take "embarrassingly parallel" [5] problems, i.e. problems which can be easily broken up into multiple smaller, independent subtasks. These subtasks can then be assigned to a magnitude of different computers which perform the computations all in parallel. In the case of SETI@home, e.g. every computational entity analyses just a small portion of the sky which can be done independently from the analysis of other portions. That being said, virtually any embarrassingly parallel problem can be accelerated this way. This includes, but is not limited to, evolutionary computation, metaheuristics, simulation of particle physics, numerical weather prediction, CNF solving and crypto-currency mining.

However, volunteer computing relies on altruistic participants who donate their computational power to one or more research projects and who are most likely found in the scientific community only. Furthermore, these volunteers typically are very picky about which project they contribute their resources to; it is very common that a project must provide some scientific impact or otherwise gain the interest of volunteers in order to attract any meaningful amount of computational resources. However, there are many use-cases that do not provide any scientific advances but which are still in need of immense computational resources. In this paper, we address this issue and propose a distributed computing system that builds around Blockchain technology and rewards users with crypto-currency payments for sharing their computational resources with scientific projects. The proposed approach creates a non-altruistic incentive for the broader general public to contribute computational resources and attracts users outside the boundaries of scientific communities. This allows for the system to be accessible not only for scientifically meaningful computations but also for, e.g., computationally intense computations in the corporate world. In the remainder of this document, we will describe the end-to-end workflow in detail from the user's point of view.

2. Paradigm

Our goal was to create an open-source Blockchain-driven middle-ware system for the distributed computation of embarrassingly parallel computational problems. A great focus has been put on making the platform sustainable, i.e. building a platform that does not depend on any person, central infrastructure which is required to be maintained, or any continuous funding. We believe that Blockchain technology holds great potential to achieve this goal; it offers a solid layer of robustness as it is a highly redundant decentralized system and allows us to rethink the way we design systems from the ground up. In particular, we leverage this new technology to build a "better" version of BOINC on-top of a decentralized digital ledger across an arbitrarily scalable and fault resistant network of computers without the need for a central authority ensuring fair play with all participating entities acting as independent equal agents in an open, free and fair environment. While censorship, a monopoly on the fair market value of a certain amount of computational power, and favoring certain (more pleasant) projects is theoretically possible in centralized systems, this no longer is the case in such decentralized environment.

3. Terminology

In this paper, the term "scientist" is used in a broader sense and refers to any entity that seeks to use the computational resources available in the network in order to solve some complex computational task which is referred to as a "job". Entities who contribute their computational resources in exchange for Crypto-currency payments are called "workers".

There are two ways workers can earn units of the underlying Crypto-currency: they can be either rewarded for their on-going work on a problem or for finding an actual solution to the problem. The on-going commitment is rewarded by allowing workers to frequently submit partial (yet incorrect) solutions that meet certain criteria. These solutions are called "PoW submissions" and the underlying challenge is called "PoW". The term "bounty" in this study will refer to the latter of the two pathways and described the actual solutions to the problems. Respectively, the amounts workers are paid for each of these submissions are termed "PoW reward" and "bounty reward".

4. End-to-end Work Flow

4.1. Work Creation

The entire workflow typically starts with scientists composing jobs which contain all the logic to solve for the actual problem as well as information regarding the rewards being offered to incentivize participation in solving the problem. To be more precise, scientists express their algorithmic problem using *ePL*, a *DSL* (domain specific language) particularly designed for this use-case, which is highly parallelizable and allows to define specific criteria that determine when a solution to their problem has been found. While some may argue that introducing a new programming language may limit the adoption of this decentralized platform, we'd like to point out that it is relatively straightforward to overlay an *IDE* (integrated development environment) which allows scientists to code in familiar languages such as Python, C, and Java while the IDE converts their code to conform to the requirements of ePL.

That being said, workers that find actual solutions – or bounties – are rewarded with a "bounty reward" by the scientist paid in units of the underlying Crypto-currency. These bounty rewards are set by the scientist when the job is created and should be ideally set to an amount that attracts participation in running their job. As the network grows, competition among scientists to attract participation in running their job increases; therefore, scientists will want to calibrate their bounty rewards to a fair market value to ensure interest in their job.

While bounties are the primary incentive to attract computation nodes to work on a job, there is a risk that nodes could be working on jobs where no bounty solution even exists. This could be due to an intentional malicious act by the scientist, or simply a bug in the job's code. To mitigate this risk, scientists will be required to additionally provide PoW rewards. *PoW* is defined as moderately hard tasks which are easily verifiable and which are randomly found at a certain rate, regardless of how hard or easy the underlying task is, in order ensure workers stay motivated. PoW rewards should be calibrated to roughly match the average electricity cost of running a computational node in order to alleviate any concerns of participants that they could potentially lose money by running the job.

4.2. Working for Bounties

Jobs can be generally understood as programs which take a pseudo-random input, run a logic coded by the scientist and output whether this particular input constitutes a PoW, a bounty or nothing. To give a better understanding, such logic could be for example a variant of the Travelling-Salesman Problem[4] with the random input translating to one particular solution candidate. Furthermore, the bounty criterion could be a check whether that particular solution candidate – in this example, a Hamiltonian path from a start to a destination point – has a total cost which is below a certain threshold.

Workers are searching for bounty solutions by continuously generating new pseudo-random inputs to the function until they find an input that yields in a bounty solution. These random inputs used by the logic will need to be calculated by the computation node using a methodology we called *personalizedInts*. Rather than simply using a random number generator to create random inputs for each computation node, *personalizedInts* uses publicly known node and job-specific *values* such as the worker's public key, the job's ID, the block ID of the block the job was announced in and some random noise that the worker can randomly pick. This is necessary to ensure each solution can be directly tied back to the node that found the bounty solution in order to prevent certain types of solution stealing attacks. If a node were to intercept a solution and submit it as their own, their public key would be different resulting in a different pseudo-random input, which would likely result in a solution that does not meet the threshold required by the job author.

When a worker finds a bounty solution, they will submit the values used to generate the random inputs to the calculation as well as the output of the calculation as defined by the job author. If the bounty solution can be verified by the rest of the network, the worker is paid the bounty reward by the scientist.

4.3. Working for PoW Solutions

PoW Solutions can be found at almost no additional cost after performing a full evaluation of the scientist's code while checking if a certain pseudo-random input constituted a valid bounty solution. The POW task is simply to perform a *MD5 hash* of the results of each run of the job's code and compare that to a *PoW target hash* set by the network. If the computational node's run produces a hash less than the target, they are eligible for a POW reward. When a PoW solution is found, the inputs and calculation outputs are submitted, similar to submitting a bounty solution. The network recalculates the PoW target hash once per block with a goal of ten PoW rewards per block (with a maximum of 25 per block) averaged over all currently running jobs. That is, given a block-time of 60 seconds, the network tries to converge towards a state where all jobs, in total, payout ten PoW rewards per minute. Similar to the case outlined above, workers get rewarded the PoW rewards for every PoW submission that the network can positively validate.

4.4. Validation of Results

The validation is the most important part of the system. However, while its absolutely critical to perform this validation step, many calculations are far too complex and time-consuming to perform in a timely manner across the entire network. While calculations, which are carried out on the worker's hardware, can run for a while without bothering anyone, that cannot be said for the verification part which every single node in the network

must be able to execute in a reasonable amount of time. Hence, there is a very strict requirement on how complex a verification can be. If a scientist happens to have a too complex algorithmic logic, he has the option to provide an alternative simplified verification logic that checks some key variables from the solution to ensure the results were valid and at the same time stays within the permitted boundaries.

During the verification process – for both bounty and PoW submissions – it has to be ensured that the solution was actually found by the node submitting it. This part is simple: the pseudo-random input for the verification is derived from the values the worker has submitted. Since these values contain the workers reported public-key, it can be easily verified that the public-key in the values actually matches the worker's public key that tried to claim credit for that solution. In the case of a PoW solution, we additionally need to ensure that the computation node is actually working on the calculation coded in the job and not just searching for MD5 hashes below the target. This is accomplished by requiring PoW solutions to include the same data required for a bounty solution, plus the MD5 hash that the computation node found while running the job. Because the verifying nodes now have all the data required to verify a bounty along with the PoW hash, they can simply run the bounty verification logic described above and then apply an MD5 hash to the outputs. This will allow the node to quickly verify that the submitting node is valid as well as confirming that the submitted POW hash does, in fact, correspond to the output from running the actual job.

It is felt that the proposed verification steps offer a robust solution that can deter the most common types of attacks that would be expected to be encountered in this type of setting.

5. Benefits of an intermediary DSL

5.1. Handling large numbers of platforms

While it is expected that the bulk of platforms contributing computational resources will comprise traditional x86 and x86-64 architecture CPUs, we focused on designing the system in a way that virtually any platform is supported, e.g. GPUs running OpenCL or Cuda and FPGAs which operate using logic gates instead of a sequential stream of byte-code instructions. Our DSL provides top-down high-level capabilities to describe algorithms in a platform-independent manner using a very generic set of operators. This DSL then allows workers to locally translate the logic into platform-specific code and locally compile it for the desired target architecture.

5.2. Preventing Malleability

A primary focus of the DSL is to ensure that all jobs terminate gracefully in a timely manner and post no thread to the worker's platform whatsoever.

The DSL consists of a pre-processor that checks that any submitted job conforms to the standards of the proposed language, as well as a runtime component to prevent crashes of the program. Overflows, division by zero and other illegal statements identified by the pre-processor will cause a job to be rejected by the network. Also, because many of these illegal statements won't manifest themselves until the code is run, the DSL contains a runtime component that continuous checks for illegal conditions. If such a condition is encountered, the offending step in the logic is bypassed rather than causing the job to crash.

To prevent jobs from running endlessly, traditional `FOR`, `WHILE`, and `DO` loops as well as `GOTO` statements have been removed from the language. Instead, we have provided a `REPEAT` statement that requires scientist to provide an upper limit for the number of iterations which allows the pre-processor to estimate the maximum computational effort it will take to execute the loop. That allows for a precise *WCET* (worst-case execution time) estimation of the entire program logic which would be not the case for deeply nested, highly-pipelined traditional loops. Now, to ensure the logic runs in a timely manner, the pre-processor estimates the amount of computation effort using WCET analysis; only programs with a verification (and execution) WCET below a certain threshold are allowed to be submitted to the Blockchain in order to prevent clogging down the entire network.

Additionally, the DSL does only allow to use an isolated, restricted and highly limited memory space which prevents malicious code to both eavesdrop on any other data and attack the worker's platform by allocating more resources on the target platform than are available.

6. Summary

In this paper, we have described the general principles of public-resource volunteer-computing paradigms and identified a few aspects that could be improved upon. Hence, we have proposed a slightly adapted version of these traditional approaches, XEL, that leverages Blockchain technology in order to allow scientists to create an additional incentive for workers to share their computational resources. These adaptations attract a number of people to participate even if they have no further interest in the scientific projects they are working on – a preliminary assumption in traditional volunteer-computing. We have given a brief overview of all important aspects of the end-to-end workflow, beginning with the creation of new jobs all the way to the verification of solutions that workers find. While very similar to traditional volunteer-computing, a number of crucial changes to the way jobs and solutions are processed had to be made since the monetary incentive is likely to attract all sorts of different attacks on the system. The exact details of this approach are beyond the scope of this high-level paper but will be covered in a follow-up full paper version.

References

[1] David P Anderson. "Boinc: A system for public-resource computing and storage". In: *proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. IEEE Computer Society. 2004, pp. 4–10.

[2] David P Anderson et al. "SETI@ home: an experiment in public-resource computing". In: *Communications of the ACM* 45.11 (2002), pp. 56–61.

[3] Matt Cutts. "An introduction to the gimp". In: *Crossroads* 3.4 (1997), pp. 28–30.

[4] Robert L Karg and Gerald L Thompson. "A heuristic approach to solving travelling salesman problems". In: *Management science* 10.2 (1964), pp. 225–248.

[5] Wikipedia. *Embarrassingly parallel — Wikipedia, The Free Encyclopedia*. `http://en.wikipedia.org/w/index.php?title=Embarrassingly\%20parallel&oldid=830119483`. [Online; accessed 17-May-2018]. 2018.

[6] Wikipedia. *Volunteer computing — Wikipedia, The Free Encyclopedia*. `http://en.wikipedia.org/w/index.php?title=Volunteer\%20computing&oldid=840807707`. [Online; accessed 17-May-2018]. 2018.