# Distributed Password Cracking in a Hybrid Environment

**Radek Hranický, Lukáš Zobal, Vojtěch Večeřa, Petr Matoušek**

ihranicky@fit.vutbr.cz, xzobal00@stud.fit.vutbr.cz, xvecer18@stud.fit.vutbr.cz, matousp@fit.vutbr.cz

Faculty of Information Technology
Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic

## Abstract

For forensic experts, encrypted data nowadays represent one of the biggest challenges. With criminal suspects unwilling to surrender their passwords, the only way to obtain the encryption key is by password disclosure or password cracking. Modern cryptographic techniques are being enhanced to provide the maximum level of security making it impossible for a single man with a computer to crack the password within a meaningful time. General-purpose computing on graphics processing units (GPUs) often accelerates the entire process. Nevertheless, a single-machine GPU cracking has still its limits. Thus, the only way of achieving the desired amount of computational power is distributed cracking.

Not in every case the forensic investigators possess a dedicated computer cluster powerful enough to serve their needs. Complex problems may require merging multiple clusters together as well as the use of general-purpose hardware to get as much computational power as possible. The computers may be even located in geographically separated areas, e.g. multiple corporation branches, requiring a solution for secure and reliable interconnection and control of the machines.

Using Berkeley Open Infrastructure for Network Computing (BOINC) framework together with our specialized modules, we designed a solution for distributed password recovery in a hybrid environment including, but not limited to, individual general-purpose CPU nodes, GPU nodes, and specialized CPU/GPU clusters. The solution is feasible for a variable number of nodes in an untrusted and unstable environment, by offering proper adaptivity and robustness.

In the paper, we provide the experimental results of distributed cracking in a hybrid CPU/GPU environment. The experiments include both exhaustive search and dictionary attack. We analyze the results of different approaches using several types of hardware. Our goal is to provide a survey comparing possible attack techniques and connecting them with the most-fitting cases. The results may help forensic investigators to choose a proper method for each case.

**Keywords:** distributed computing, digital forensics, cryptography, password cracking, BOINC

## 1 Introduction

With the increasing number of digital devices and the growing capacity of their data carriers, the users still have more space to store their personal data. This fact may turn into a problem if a user becomes a suspect of a crime and the data carriers are seized for forensic analysis. Nowadays, the amount of data is the major challenge for forensic experts since the bigger the obtained data is, the harder it is to find the desired clues. Captured material may exceed dozens of terabytes, making it impossible for a single man with a computer to find the desired "needle in a haystack".

Another challenge comes if the forensic experts have to deal with a piece of data which is encrypted to prevent unauthorized access. For many media types, like encrypted documents, archives, or disk partitions, the most frequently-used formats have standardized cryptographic procedures, and the cryptographic algorithms are well-known. However, even if the used encryption algorithm is known, in order to obtain the original data, we need an encryption key or a password from which the key can be derived. Some countries enacted the key disclosure law requiring the criminal suspects to surrender their passwords conditioned by a higher penalty in a case when suspects are unwilling to cooperate. However, even with this law, the disclosure of the cryptographic keys is not guaranteed, and thus the only option is the password cracking, also frequently called the password recovery.

The password cracking is a method of obtaining passwords by force. Based on the concrete attack type, we try every possible password from the assumed password space until eventually the correct password is found, or all possibilities are exhausted. The password space can be reduced by every known information about the password, e.g. its maximum length, used characters, or the language if the password is an existing word. The size of the password space tremendously impacts the complexity of the entire process. Another important factor is the computational complexity of the cryptographic algorithms required to verify each password.

Nowadays, the password cracking is becoming more important. The law enforcement agencies (LEA) are reporting an urgent need for efficient cracking tools, and the number of requests for more specific features is increasing. Such requests are coming not only from the Czech LEAs, but from different agencies around the world, and also various companies offering forensic services to their national LEAs.

Even though the cracking process is often computationally complex, it can be easily parallelized by verifying several different passwords at a time. For parallel computing, we can use multiple CPU threads, however, on commodity CPUs, the number of threads running consequently is limited by the number of physical cores, which results in verifying a limited number of passwords at the same time. For higher acceleration, we can use general-purpose computing on graphic processing units (GPGPU) which allow the use of GPUs to compute cryptographic algorithms as well. With common algorithms and high-end GPUs, usually thousands of passwords can be verified concurrently.

Despite the fact that using GPGPU can make the process multiple times faster, the number of GPUs possibly attachable to a single motherboard is limited. Thus, for big password spaces or complex algorithms, it may be impossible to find a correct password in a meaningful time. In such cases, the only solution is to distribute the computing to multiple nodes. The concrete technology to be used for distributed computing relies on the type of distributed environment. In the "best case", the LEA possesses a cluster of identical GPU nodes connected by fast links creating a powerful computational platform. However, the more the network differs from this optimal state, the more complex strategies we need to preserve the efficiency of the cracking process.

### 1.1 Contribution

In the paper, we discuss the common issues that affect the performance of distributed computing. For various use-cases, we suggest solutions for an efficient use of resources. Using both analytical and experimental approach, we underline the aspects of distributed dictionary attack and demonstrate them on real-world use-cases. Our focus also aims at a practical demonstration of adaptive scheduling of tasks in a distributed computing environment.

### 1.2 Structure of the Paper

The paper is organized as follows. Section 2 presents related work. Section 3 discusses the advantages of using GPGPU and possible methods of distributed computing. In section 4, we define the term "hybrid environment" and look how it affects the password recovery process, while section 5 shows the experimental results of our software in such environment. In the end, section 6 concludes the paper.

## 2 Related Work

In the area of password cracking, previous research discovered a high potential of distributed cracking using GPGPU. Single-machine GPU-computing, however, has its limits making the distributed computing in many cases necessary to find the correct password in a reasonable time. Kim et al. proposed the basic idea of the controller-worker model and experimentally proved its use by cracking PDF 7.0 files on 88 nodes containing 342 GPUs in total. For numeric 12-character password, the GPU acceleration lowered the cracking time from 7 days to 15 hours [7].

Apostal et al. experimented with distributed dictionary attack using MPI and CUDA technologies and compared the alternatives of dividing password databases between different nodes, proving the efficiency of the divided dictionary algorithm [8]. Zou et al. proposed a model for distributed password recovery using brute-force and dictionary approaches, proclaiming that writing "optimal codes" for low-level cracking may not be sufficient, and proper distribution algorithms have to be used [9].

Comparing existing single-machine cracking software, Hashcat[1] has shown to be probably the most powerful tool for GPU-accelerated password recovery. Hashcat team has taken the 1st place on CMYC[2] "Crack me if you can" contest on DEFCON 2012. Despite its performance and popularity, it does not natively support distributed computing. Since the comparison with Wrathion tool in 2015 [2], Hashcat developers have made a long way regarding the optimization of GPU kernels and adding support for new encrypted media formats. There are, however, various less-or-more known distributed password recovery solutions. From a wide range of commercial tools to few open-source tools most of them based on Hashcat.

In our previous research, we have studied different technologies usable for distributed computing detecting that MPI can be a smart solution for a dedicated cluster of similar nodes in a trusted, stable environment. However, for a hybrid environment with a variable number of nodes interconnected by an unstable, untrusted network like Internet, many of the existing technologies are not sufficient. For such environment, a lot of existing tools either lack a support or use proprietary protocols. We detected the flexibility and robustness make BOINC[3] framework a suitable technology for use-cases which involve hybrid, unstable, or untrusted environment. Thus, we proposed the Fitcrack[4] system for distributed password recovery, on which, we have proven the practical usability of the BOINC-based design [1].

# 3 From Single-Machine CPU to Distributed GPU Attack

The time required for password cracking depends on the algorithms involved, on the password itself, and on the computational power we have. Using a brute-force attack, the maximum time in seconds ($t_{max}$) to find a password can be computed by the following equation:

$$t_{max} = \frac{\sum_{l=min}^{max} |A|^l}{s}$$

where *min* is the minimum password length, *max* is the maximum password length, *A* is the alphabet used, and *s* is the cracking speed in password per seconds.

For instance, cracking a ZIP archive encrypted by AES with 256-bit key with an alphanumeric password made of 8 characters, i.e. *min* = 1, *max* = 8, and *A* = {a…z,A…Z,0…9} can take over 867 years with *s* = 8112 which is the cracking speed achieved using Elcomsoft[5] Advanced Archive Password Recovery 4.54 on Intel(R) Core i7 CPU 920.

## 3.1 Technological Limits for Single-machine Computing

No matter if we use CPU or GPU for critical computations, or how powerful processors we assume, using a single machine provides limited computational power.

Since many digital forensics tools support multi-threading and today's CPUs usually contain multiple cores, parallelization on CPU is a basic approach for making the computation faster. Intel(R) Xeon(R) E7-8890 v4 has 24 cores which allow with hyper-threading to run up to 48 threads parallelly. Using a motherboard with multiple CPU slots can offer even more power, e.g. Supermicro X8QB6-F server board with 4 CPU slots. The number of CPU slots is, however, still limiting. Moreover, for tasks like password recovery, the CPU-only computation has proven not to be efficient enough [2].

For tasks which require highly parallel computation of a single algorithm with multiple values, GPGPU should be considered. Computing the generation of passwords, encryption and hashing algorithms on a GPU may be a wise choice for achieving speedup [3, 4, 2]. Such tasks include password recovery where basically GPU can run hundreds or thousands of parallel kernels, each of them cracking a single password. With Wrathion[6] tool, cracking

[1] See https://hashcat.net/

[2] See http://contest-2012.korelogic.com/stats.html

[3] See https://boinc.berkeley.edu/

[4] See http://fitcrack.fit.vutbr.cz/

[5] See http://elcomsoft.com

[6] See http://wrathion.fit.vutbr.cz

the ZIP encrypted by 256-bit AES using single AMD Tri-X R9 290X GPU has shown to be 30.28x faster than using Intel(R) Core i7 CPU 920. By adding a second GPU of the same type, the speed even doubled [2]. For obvious reasons, many current password recovery tools like Hashcat, John the Ripper, AccessData, or Elcomsoft software, support GPU acceleration. Using a single machine is, however, still limited by multiple aspects.

The number of PCI-e slots depends on the motherboard and is limited. High-end gaming boards can offer 4 or more PCI-e x16 slots, e.g. MSI Big Band-Marshal offers 8 PCI-e x16 slots. There are also unconventional PCI-e splitters, expansion cards, backplanes, and adapters, which can increase the number of PCI-e slots. Despite the fact that using third-party gadgets is risky, PCI-e slots cannot be extended infinitely. There is the maximum number of PCI-e lanes supported by the CPU. Even with additional PCI-e bridges, the BIOS on all commodity motherboards does not allow to access more than 64K of IOSPACE [6].

High-end GPUs are relatively big in dimensions which may lead to problems with inserting more GPUs and fitting in the computer case. This can be, however, solved by third-party extension cables and by using a custom solution of the computer casing. On full load, high-end GPUs also produce a significant amount of heat, increased by multiple GPUs mounted close to each other. This requires proper cooling system, and often a special cases. Water cooling can solve the problem, but external coolers require additional space.

Another limiting aspect is power supply unit (PSU) which, especially with the use of multiple GPUs is very limiting. The strongest consumer PSU end on 1600W (EVGA SuperNOVA 1600 G2) or 2000W (Super Flower Leadex Platinum). More power requires unconventional solutions like Add2Psu[7] enabling to stack up to 4 PSUs together. Least, but not last comes the price. It is always to consider if buying a special, or even third-party hardware is more advantageous than building multiple machines for distributed computation.

### 3.2 Multi-Machine Computing

Assuming the limits of a single-machine approach described in 3.1, a working alternative for obtaining more computational power is distributed computing. The basic idea is to make multiple physical or virtual machines cooperate on a common problem by putting their computational power together.

There are many models describing distributed systems. We can define separated models by the network topology used (centralized/decentralized, star, ring, hierarchical, etc.), or by the type of communication (synchronous vs. asynchronous) [10]. Although, it is possible to develop a custom library or protocol, e.g. based on TCP/IP and using existing networks. There are various existing solutions which implement underlaying operations and allow the developer to focus on the problem itself [1]. In our case, we assume a network with a single central node – a server, and multiple working nodes connected either by a local network, the Internet, or both.

## 4 Password Cracking in Hybrid Environment

Based on the type of an attack (brute-force, dictionary-based, rule-based, etc.) and further configuration (used alphabet, dictionary, etc.), we define a set of all possible passwords. The goal of the password cracking is to detect if the set contains a correct password and if so, report it. As we have described in [2, 1] the entire process of the password cracking is based on an iterative principle, where each iteration generates and then verifies a defined number of password(s). The computation ends when a correct password is found, or once the entire set is exhausted. In a distributed cluster, we usually have identical nodes or groups of similar nodes. A hybrid environment represents a network of different nodes, i.e. CPU-only and GPU nodes, or nodes with considerably different performance.

### 4.1 Splitting the Set of Passwords

Based on the information denoted above, we need to distribute the passwords among nodes. Which strategy we use for distribution depends on the stability and trustworthiness of the environment.

In a fully stable and trusted environment with *n* working nodes, we can split the entire set of all possible passwords into *n* subsets and assign a subset to each node at the beginning of the process. Such approach implies zero overhead except for the start and the end of the entire process where the server has to communicate with the nodes

[7] See http://www.add2psu.com

to assign tasks and read their results [1]. If nodes differ in performance, we can assign a weight to each node and split the set non-uniformly, i.e. a node with higher performance receives a larger set of passwords and vice versa.

In an untrusted environment, a different strategy has to be chosen. We do not split the entire set at the beginning; instead, we create little subsets (jobs) during the entire process and assign them to nodes dynamically. Thus, if a job fails for any reason, we lose only a small subset of passwords which is easier to assign again. In an unstable environment, we expect possible changes in the performance of nodes or even changes in the number of nodes during the process. This approach requires adaptive scheduling of jobs, where we create jobs specifically for given nodes based on their current speed and the "optimal" amount of time that we choose for computing a job. Fitcrack system uses adaptive scheduling by default [2].

### 4.2 Generating Passwords

Depending on the implementation of the cracking tool and the type of an attack, we can generate the passwords:

- **Offline** – The passwords are generated at the central node (server) before the cracking even starts, resulting in a dictionary of passwords. The dictionary can also be uploaded manually to the server. For each job, we select a part of the dictionary and distribute it to the working nodes. This leads to distributed dictionary attack (DDA). The advantage is re-usability of the dictionaries and lesser requirements to the cracking software on working nodes. A disadvantage is, in some cases, the size of the dictionary and the speed of the interconnecting link. We discuss the problem more deeply in 5.1 and 5.2.
- **Online** – As a job, we do not distribute passwords, only a range of password indexes (e.g. serial numbers). This minimizes the overhead for communication and input/output operations, however, every node must implement the password generator used. Concretely, a node must be capable of generating a unique password for each index. We use this approach in 5.3 for a brute-force attack.

## 5 Experimental Results

Since we have already analyzed the achievable speed and efficiency of both stand-alone, and distributed brute-force attack [1, 2], we now focus on a practical use of distributed dictionary attack (DDA). We also performed an experiment with an unstable environment simulated by manually loading a node with another exhaustive process. For our experiments, we used both Hashcat tool and Fitcrack distributed system based on BOINC [1].

### 5.1 The Hypothesis: An Analytical Approach on Hashcat

Unlike distributed brute-force attack, DDA is highly dependent on the speed of the communication network connecting the cracking nodes. With online (see 4.2) brute-force attack, we only need to distribute password indexes while the passwords are generated on the cracking nodes. However, with DDA, when a new dictionary (unknown to the nodes) is used, the parts of the dictionary have to be distributed to the nodes. The possible cracking speed is different for each file format and depends on the hardware and software used [2]. Thus, we assume, for higher cracking speeds and slower network connection, the nodes may spend more time by waiting for passwords than the cracking itself, while for more computationally-complex formats, the attack may be worth even with a slower network. For comparison we define a variable $x$ representing the ratio of the cracking speed to the "bottleneck" speed of interconnecting lines.

Let us have a dictionary of passwords in ASCII format with an average length of $p$ bytes including the line feed symbol (0x0A). Let us have a star-based topology where the center is an Ethernet switch with a server and $n$ working nodes attached, and let both nodes and the server be connected with a line of $s$ b/s. Ignoring the overhead of protocol headers, from the server to a single node, we can transfer the number of $c_1$ passwords per second using the following formula:

$$c_1 = \frac{s}{p \times 8}$$

With a transfer to all nodes, the best case is:

$$c_2 = \frac{c_1}{n}$$

The only possible speedup could be achieved by using a broadcast transfer which is, however, hardly applicable since transferring the same set of passwords to multiple nodes makes no sense. Let $f$ be the cracking speed of a given format. We can now estimate the ratio $x$ of the cracking speed to the link speed:

$$x = \frac{f}{c_2} = \frac{f \times n}{c_1}$$

For example, $x = 1$ signifies that the cracking speed is equal to the link speed which makes the time of the distributed cracking twice as long as the time required for a single-machine cracking with the same speed. In the optimal case, $x$ is to be as close to 0 as possible making the time required for dictionary transfer negligible.

Let $K$ be the maximum acceptable value of $x$, signifying that for formats with $x > K$, the dictionary attack is inefficient.

Let us assume the network consist of the server and a single node with 8x NVIDIA GTX 1080, then $n = 1$. Let the nodes be connected by 1 Gbps line, then $f = 1{,}000{,}000{,}000$ p/s. Let $p$ be the average password length of RockYou leaked[8] dataset, so $p = 9.75$. In this example, let the attack be considered efficient when the speed of password transfer is at least two times higher than the cracking speed which implies $K = 0.5$.

| File format | Cracking speed (f) [p/s] | x |
|---|---|---|
| MS Office < 2003 | 2,743,900,000 | 214 |
| MS Office 2003 | 2,667,600,000 | 208 |
| MS Office 2007 | 1,073,500 | 0.08 |
| MS Office 2010 | 535,400 | 0.04 |
| MS Office 2013, 2016 | 70,884 | 0.005 |
| PDF 1.1 – 1.3 | 3,014,200,000 | 235 |
| PDF 1.4 – 1.6 | 129,300,000 | 10.09 |
| PDF 1.7 (Adobe Acrobat 9) | 22,938,300,000 | 1,789 |
| PDF 1.7 (Adobe Acrobat 10 – 11) | 256,200 | 0.02 |
| RAR v3 | 239,200 | 0.02 |
| RAR v5 | 290,300 | 0.02 |
| WinZIP with AES | 8,463,300 | 0.66 |
| 7-ZIP | 60,750 | 0.004 |

Table 1: The cracking/line speed ratio of different file formats

Assuming the benchmark[9] of Hashscat 3.30, Table 1 shows the cracking speed, and the computed ratio (x) of the cracking speed to the line speed. We may see that for less secure formats like MS Office 2003 and older, $x$ is many times higher than $K$ making the dictionary attack practically unusable. In contrast, formats like MS Office 2007 and newer, or RAR are much more computationally intensive, making the 1 Gbps line more than sufficient for a practical application of DDA.

Based on the previous observations, we suppose DDA is worth only for cracking newer well-secured file formats. We also assume the actual efficiency of the attack is inversely proportional to $x$.

[8] See https://wiki.skullsecurity.org/Passwords

[9] See https://gist.github.com/epixoip/a83d38f412b4737e99bbef804a270c40

## 5.2 The Efficiency of Distributed Dictionary Attack

To prove our hypothesis from 5.1, we demonstrate the efficiency of Fitcrack distributed system in a hybrid network. Assuming the formats differ in the cracking speed, for each format we use a modified RockYou dataset and a suitable password to be cracked in time less than 1 hour. Since our goal in this experiment is not to achieve a high cracking speed, neither to show the time required to find a password, we set the adaptive scheduling subsystem [1] to create approximately 2-minutes jobs to make the overhead more visible. Our goal is to compare the efficiency of DDA used for cracking different file formats. Based on the hypothesis, we selected some of the computationally complex formats. Our distributed network has the following configuration:

- 16 nodes with 1x Intel(R) Core(TM) i5-3570K @ 3.40 Ghz,
- 2 nodes with 1x AMD Radeon R9 Fury X,
- 1 node with 4x AMD Radeon R9 Fury X.

Page et. al. [5] defined the efficiency (Eff) of distributed computing as *the percentage of the time that processors actually spend processing rather than communicating or idling*, which can be computed as:

$$Eff = \frac{\sum_{x=1}^{N} t_x}{N \times T_{fin}}$$

where N is the number of nodes that participate in computing, $t_x$ is the real time of task processing spent by node x, and $T_{fin}$ is the final time of the entire computational process.

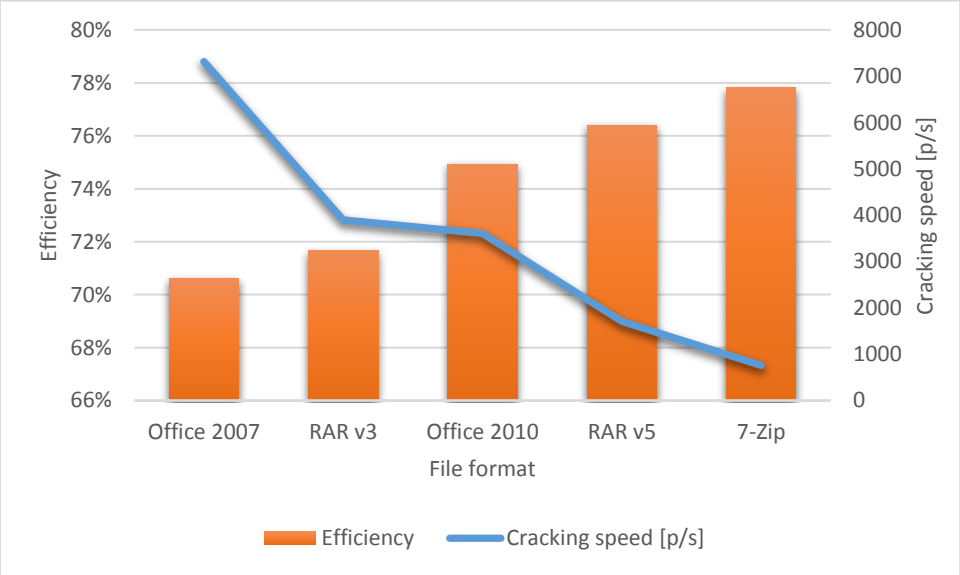


Figure 1: The relation between efficiency and speed shown on experimental results

Based on our observations from 5.1, we decided to make experiments with some of the more computationally intensive formats (where $x \leq 0.5$). Figure 1 shows experimentally measured efficiency and cracking speed. We chose MS Office 2007, RAR v3, Office 2010, RAR v5, and 7-Zip. From the chart above, we can see the relation between the efficiency and speed. The most "hard-to-crack" format is 7-Zip where the cracking speed is the lowest while the efficiency is the highest. In contrast, Office 2007 showing the highest cracking speed resulted in the lowest efficiency among all five formats. The measured values prove our hypothesis from 5.1.

## 5.3 Adaptive Scheduling in an Unstable Environment

In a hybrid environment, nodes typically differ in performance and the performance of a node may even change over time. In our previous research, we proposed an adaptive scheduling algorithm [1] which creates tasks to fit the node's actual performance and is able to handle changes in the network topology. In this section, we want to experiment with performance and show, how the same approach can be used in a hybrid environment.

To make the results comparable, we used faster CPUs and slower GPU. We were cracking PDF 1.7 revision 5 with a brute-force generator to eliminate the negative effect of dictionary transfer described in the previous experiments. Our network consisted of four nodes. We measured the cracking speeds in Table 2 by using a benchmark. In Fitcrack, the benchmark is computed automatically to detect the initial speed of each node as described in [1].

| Name | Processing unit | Measured cracking speed [p/s] |
|---|---|---|
| Node A | Intel(R) Core(TM) i5-4200U @ 1.60 Ghz | ~ 2,700,000 |
| Node B | Intel(R) Core(TM) i7-5930K @ 3.50 Ghz | ~ 14,800,000 |
| Node C | Intel(R) Core(TM) i5-3570K @ 3.40 Ghz | ~ 9,270,000 |
| Node D | AMD Radeon R5 M255 | ~ 8,800,000 |

Table 2: Nodes' processing units and measured cracking speeds

We can see that despite the high potential of GPGPU, cracking on high-end CPU like Core i7-5930K can be faster than on low-end notebook graphics like Radeon R5 M255. We analyzed the job assignment at the beginning of the cracking. In the first measurement, all nodes compute without any further intervention. Figure 2 illustrates the sizes of tasks being assigned. To make the chart uncluttered, we showed only nodes A and B, however, the course of C and D was similar to the first two nodes – only slightly shifted on the y-axis according to cracking speed in the previous table. We may see, that after the initial tasks, the size of the job became stable. The initial deviation is caused by the imprecision of the benchmark, but has, however, no significant effect on the efficiency.

In the second measurement, we show the scheduler behaves when the performance of the node changes in time. The results are shown in Figure 3. At marked time points, we added extra load to some threads of node B which caused its performance to become lower resulting in a visibly longer time required to process the task. From longer time required to finish a job, the adaptive scheduling algorithm detected a drop in the node's performance. And thus, the algorithm assigned a smaller job. Once the load was removed, the algorithm reacted again by increasing the size of a job.

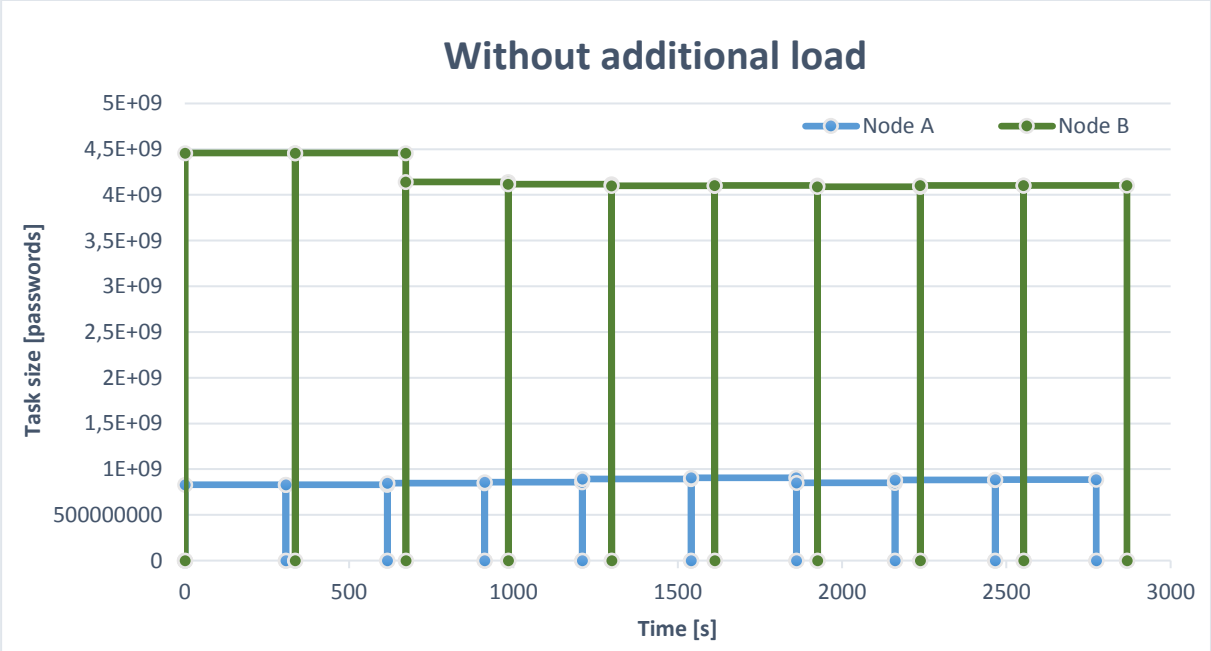


Figure 2: Task assignment without additional load

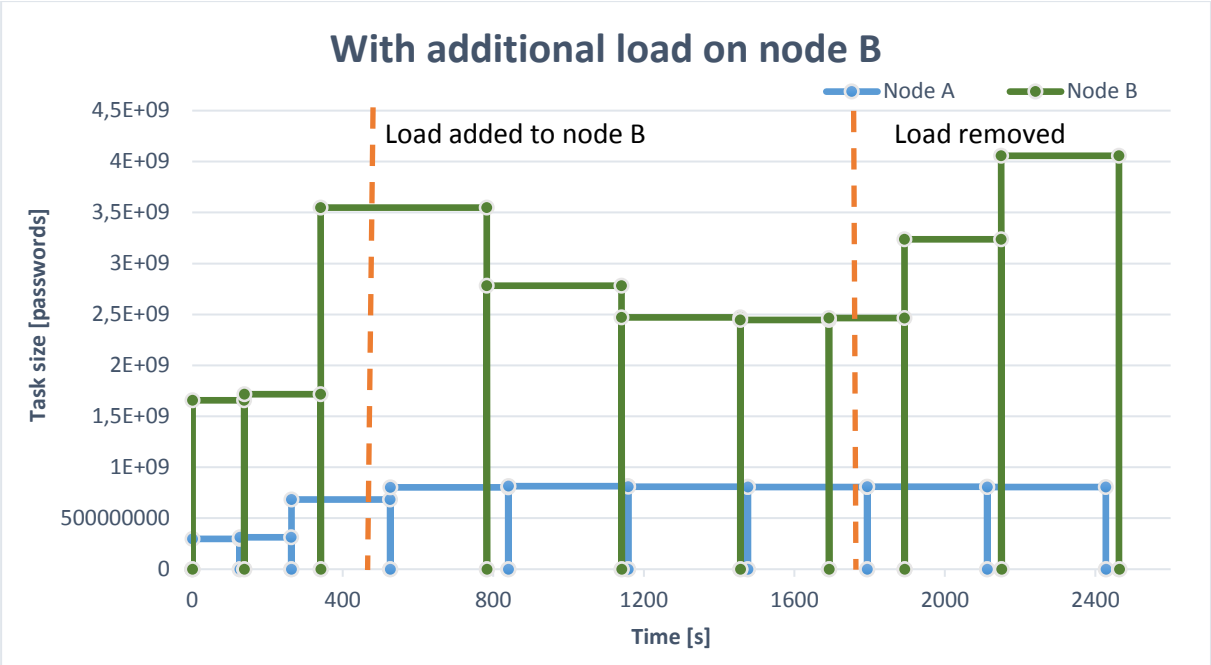


Figure 3: Task assignment with additional load

We can state, that the adaptive algorithm behaves correctly even in a hybrid environment where the node's performance changes. Since we used brute-force with online generating of the passwords, the overhead was low and we cannot see the gaps between the tasks.

# 6 Conclusions

Distributed password cracking definitely has potential to help achieve higher performance. However, different distributed networks require different strategies for distribution of the input data and scheduling tasks. For cracking in a hybrid environment, the software has to be much more adaptive than in a stable, trusted cluster of dedicated nodes with similar performance.

A practical use of the distributed dictionary attack depends on the file format, network speed, and performance of nodes. While for the computationally complex formats like 7-Zip or Office 2010 a dictionary attack does not represent a problem, for the file formats enabling higher cracking speeds the efficiency is highly limited by the speed of a network connection.

In the future, we want to enhance Fitcrack distributed system by adding the support for Hashcat tool in order to achieve maximum performance while preserving the adaptivity and flexibility of the current design. We also want to focus on intelligent password generators which employ machine learning to obtain knowledge from existing texts and dictionaries. By using various heuristics based on probability and statistics, the generators create passwords which would be more likely chosen by a human being and thus reduce the number of passwords that need to be verified.

# References

[1] R. Hranický, M. Holkovič, P. Matoušek, and O. Ryšavý. On Efficiency of Distributed Password Recovery. *The Journal of Digital Forensics, Security and Law*. 2016, vol. 11, no. 2, pp. 79-96. ISSN 1558-7215.

[2] R. Hranický, P. Matoušek, O. Ryšavý, and V. Veselý. Experimental Evaluation of Password Recovery in Encrypted Documents. In: *Proceedings of ICISSP 2016*. Roma: SciTePress - Science and Technology Publications, 2016, pp. 299-306. ISBN 978-989-758-167-0.

[3] T. Murakami, R. Kasahara, and T. Saito. An implementation and its evaluation of password cracking tool parallelized on GPGPU. In: *10th International Symposium on Communications and Information Technologies. Oct. 2010*, pp. 534-538.

[4] D. Apostal, K. Foerster, A. Chatterjee, and T. Desell. Password recovery using MPI and CUDA. In: *Proceedings of 19th International Conference on High Performance Computing*. Dec. 2012, pp. 1-9. ISBN 978-1-4673-2371-0.

[5] Page, A. J. and Naughton, T. J. (2005). Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing. In: *Proceedings of19th IEEE Symposium on Parallel and Distibuted Processing*, pp. 189a. ISBN 0-7695-2312-9.

[6] Bob Hartley. PCI Express Expansion Limitations. Tech. rep. Concurrent Computer Corporation, Feb. 2013.

[7] Keonwoo Kim. Distributed password cracking on GPU nodes. In: *7th International Conference on Computing and Convergence Technology*. Dec. 2012.

[8] Apostal, D., Foerster, K., Chatterjee, A., & Desell, T. (2012, Dec). Password recovery using MPI and CUDA. In: *Proceedings od 19th Conference on High Performance Computing*, pp. 1-9.

[9] J. Zou, D. D. Lin, and G. C. Mi. A universal distributed model for password cracking". In: 2011 *International Conference on Machine Learning and Cybernetics*. Vol. 3. July 2011, pp. 955-960.

[10] Ajay D. Kshemkalyani and Mukesh Singhal. Distributed Computing: Principles, Algorithms, and Systems. 1st ed. New York, NY, USA: Cambridge University Press, 2008. ISBN: 0521876346.