Bitcoin Throughput Analysis

Presented by:

Tal Tzafrir – 200944007

Itzhak Solomon – 201522315

Supervisor:

Alexander Manuskin

NSSL Lab

Winter 2018/9

Table of Contents

Introduction

Bitcoin has been a growing phenomenon ever since it first appeared in 2009. One of the main problems with bitcoin, is how to optimize its usage and how to receive greater throughput in a more efficient manner. We have long been interested in the subject but never really understood much about it. There are a lot of disagreements on several subjects regarding the bitcoin Blockchain, and we set out to explore the subject to try and find ways to improve the manner in which bitcoin works today.

Main objectives

To build a platform that will enable its user to change parameters of the bitcoin blockchain and measure their effect on the throughput of the system, as represented by the amount of time it takes for a single miner to send a block to its peers and receive confirmation of the verification of the block.

Side objectives

* Learn more about bitcoin.
* Learn about blockchain technologies in general and in bitcoin specifically.
* Learn about and utilize AWS.
* Improve our skills in python development.

Existing solutions

None to our knowledge, but there is academic material on the subject.

Project tools

* “Bitcoin core” bitcoin client.
* AWS – EC2 services.

Programming languages

* Python 3.7.
* C++.

Primary project stages

1. Research and learn about bitcoin and blockchain.
2. Download and experiment with bitcoin core client.
3. Writing python script to set up a number of bitcoin core nodes locally, and allow changing of blockchain parameters.
4. Modifying the python script to set up nodes in remote instances in AWS.
5. Change parameters of blockchain and time block dissemination.
6. Analyze results.

Blockchain

In order to understand what bitcoin is and how it works, we need and understanding of blockchain technology in general, and specifically some of its core concepts.

A blockchain is a growing list of records, called blocks, which are linked using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data (generally represented as a Merkle tree).

By design, a blockchain is resistant to modification of the data. It is "an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way". For use as a distributed ledger, a blockchain is typically managed by a peer-to-peer network collectively adhering to a protocol for inter-node communication and validating new blocks. Once recorded, the data in any given block cannot be altered retroactively without alteration of all subsequent blocks, which requires consensus of the network majority. Although blockchain records are not unalterable, blockchains may be considered secure by design and exemplify a distributed computing system with high Byzantine fault tolerance. Decentralized consensus has therefore been claimed with a blockchain.

Blockchain was invented by a person using the name Satoshi Nakamoto in 2008 to serve as the public transaction ledger of the cryptocurrency bitcoin. The identity of Satoshi Nakamoto is unknown. The invention of the blockchain for bitcoin made it the first digital currency to solve the double-spending problem without the need of a trusted authority or central server.

Bitcoin

Bitcoin (₿) is a cryptocurrency, a form of electronic cash. It is a decentralized digital currency without a central bank or single administrator that can be sent from user to user on the peer-to-peer bitcoin network without the need for intermediaries.

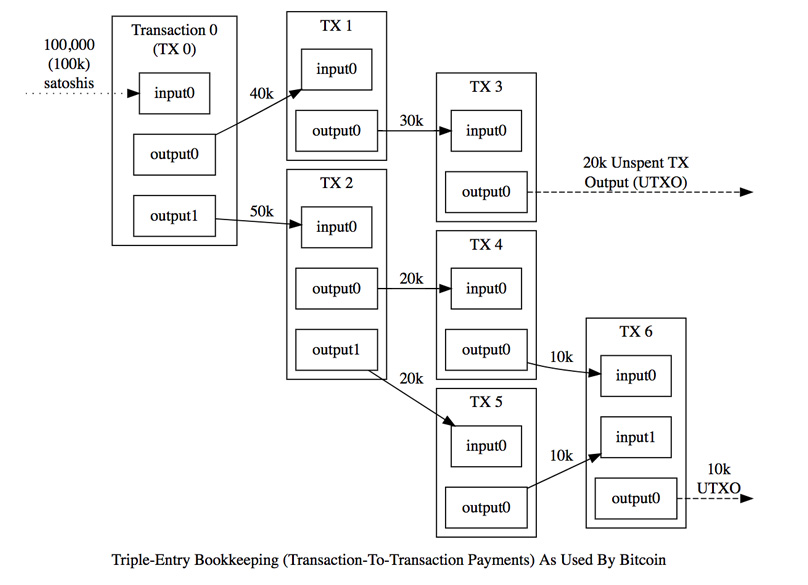
Transactions are verified by network nodes through cryptography and recorded in a public distributed ledger called a blockchain. Bitcoin was invented by an unknown person or group of people using the name Satoshi Nakamoto and released as open-source software in 2009. Bitcoins are created as a reward for a process known as mining. They can be exchanged for other currencies, products, and services. Research produced by the University of Cambridge estimates that in 2017, there were 2.9 to 5.8 million unique users using a cryptocurrency wallet, most of them using bitcoin.

Block

Transaction data is permanently recorded in files called blocks. They can be thought of as the individual pages of a city recorder's recordbook (where changes to title to real estate are recorded) or a stock transaction ledger. Blocks are organized into a linear sequence over time (also known as the block chain). New transactions are constantly being processed by miners into new blocks which are added to the end of the chain. As blocks are buried deeper and deeper into the blockchain they become harder and harder to change or remove, this gives rise of bitcoin's Irreversible Transactions.

UTXO

The fundamental building block of a bitcoin transaction is a transaction output. Transaction outputs are indivisible chunks of bitcoin currency, recorded on the blockchain, and recognized as valid by the entire network. Bitcoin full nodes track all available and spendable outputs, known as unspent transaction outputs, or UTXO. The collection of all UTXO is known as the UTXO set and. The UTXO set grows as new UTXO is created and shrinks when UTXO is consumed. Every transaction represents a change (state transition) in the UTXO set.



Bitcoin Core

Bitcoin Core is free and open-source software that serves as a bitcoin node (the set of which form the bitcoin network) and provides a bitcoin wallet which fully verifies payments. It is considered to be bitcoin's reference implementation and is the most used implementation by a large margin. Initially, the software was published by Satoshi Nakamoto under the name "Bitcoin", and later renamed to "Bitcoin Core" to distinguish it from the network. For this reason, it is also known as the Satoshi client. We ran or bitcoin network using regtest mode. For situations where interaction with random peers and blocks is unnecessary or unwanted, Bitcoin Core’s regression test mode (regtest mode) lets you instantly create a brand-new private block chain with the same basic rules as testnet—but one major difference: you choose when to create new blocks, so you have complete control over the environment.

System overview

The main parameters who’s effect on block dissemination time that we are testing in our project are (and which can be passed as arguments to the script):

* How much of the UTXO set will be stored in the cache.
* Block size.
* Network topology.

All of which can be tested on different amounts of nodes (can be passed as parameter).

In our project we used one main python script named “make\_setup\_test”, which uses one other auxiliary script named “create\_starting\_blockchain”. For the timing of block dissemination we use two more scripts named “block” and “server”. We will go into detail on each one of them. Running the system will be explained on page blaaaaaaaaaaaaankkkkkkkkkkkkkkkkk.

Parameters of interest

UTXO set

The first parameter that we wanted to check is the size of the UTXO set that will be saved in the cache. We controlled this parameter by using the -dbcache flag when running a bitcoin core node. This flag determines how much of the memory will be allocated for the use of the UTXO set. Along with this flag, when running the main script, one of the parameters passed is the UTXO set size in percentage, out of the fixed 4 MB (actually 1.8). For example, passing 50 as the parameter will set the UTXO set size to 0.9 MB.

Block size

The second parameter we checked was the size of each block in the block chain. This parameter was controlled by setting the block size in the bitcoin core source code to 16 times the original block size (original is 1MB), and then controlling the actual block size in our code when constructing the blockchain. We control it by receiving the desired block size as an argument for the script (of course up to 16MB).

Network topology

The third parameter that we checked was the topology of the network, or how all of the nodes are connected to each other. This too is decided by passing an argument the main script, and we can choose between three choices:

* Fully connected graph (clique).
* Statically random connections: nodes are connected randomly to 4 other nodes but using the same seed every time (meaning we will get the same topology for each run of the same parameters).
* Dynamically random connections: nodes are connected randomly to 4 other nodes.

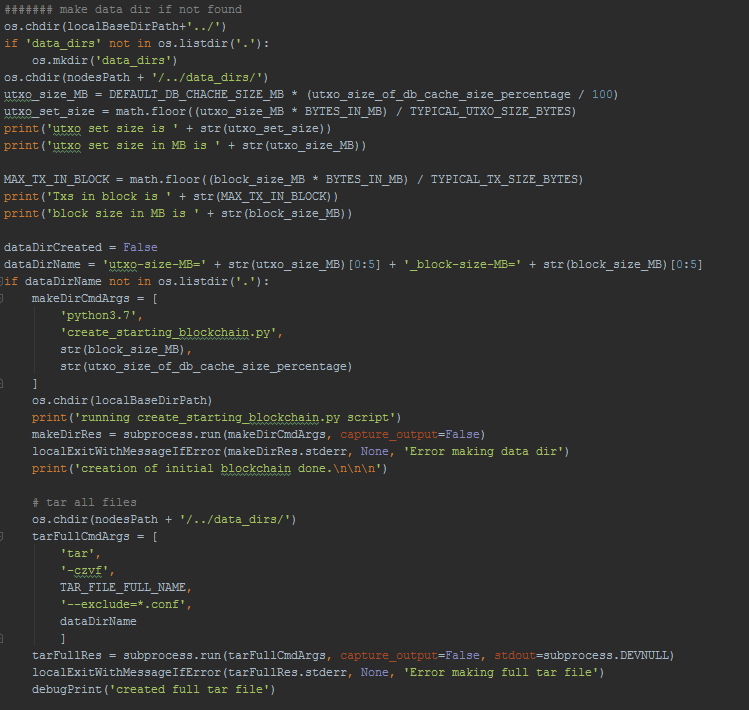
In both random options we have taken measures to make sure that the graph is connected (connected each node to its numbered successor and the 3 other nodes are chosen randomly).

Script – make**\_**setup**\_**test

This is our main script and is in charge of setting up and running our system and testing the block dissemination time.

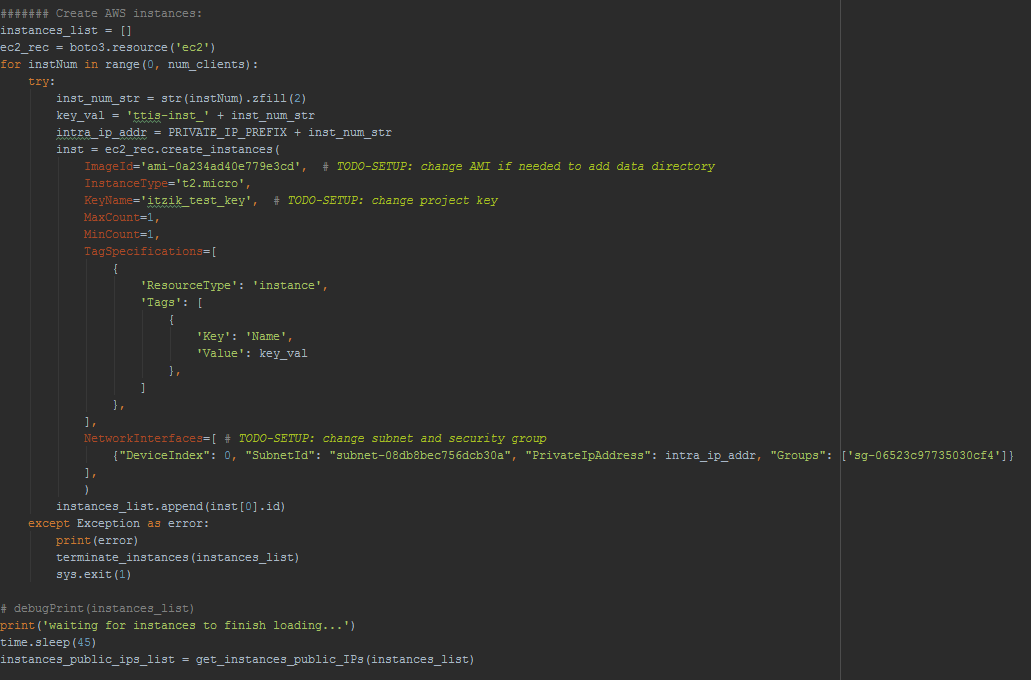
Creating data directories

The first thing that the script does is create a data directory (the main working directory for each node). It does so by calling the script “create\_starting\_blockchain” which we will cover later. The data is then zipped and ready to be sent to remote AWS instances.



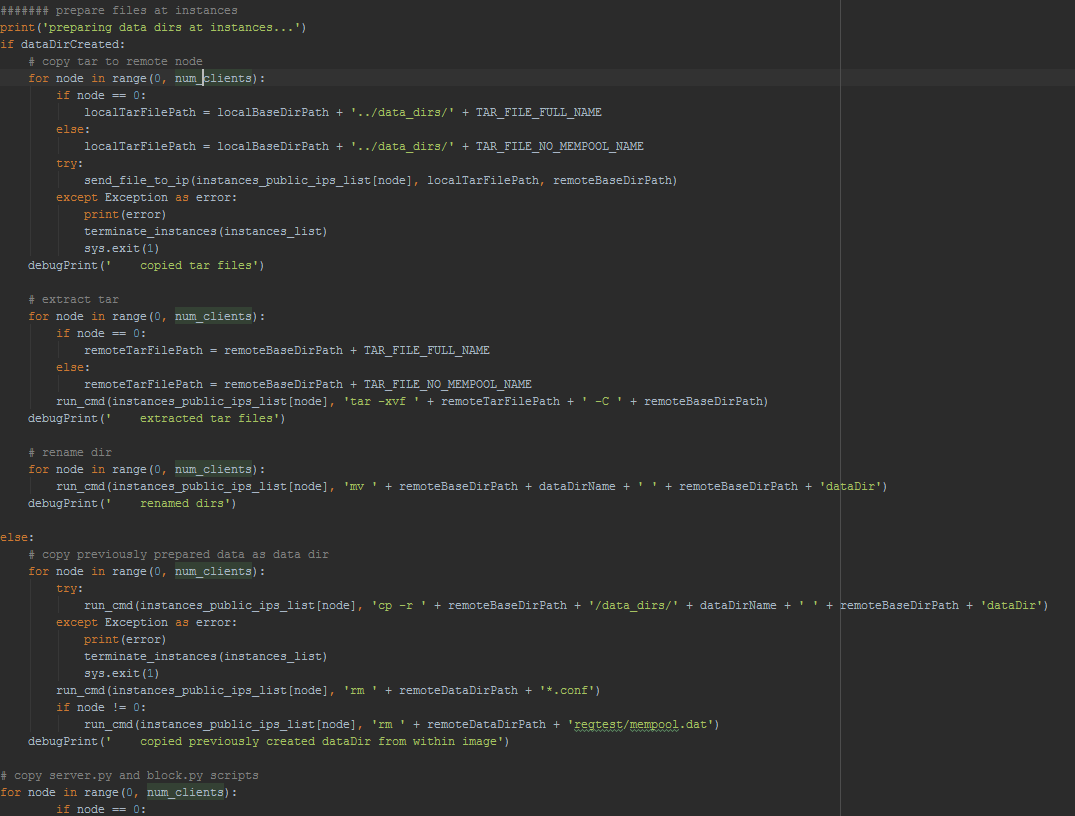
Setting up AWS

Next, the script sets up and runs a (predefined) number of AWS instances. The instances are configured to use a pre-built ami as an installation image so that everything needed by the script will already be set up. The user enters his credentials in specific configuration files so as to run the instances on their AWS account (how to do so will be covered in the “how to run” section).



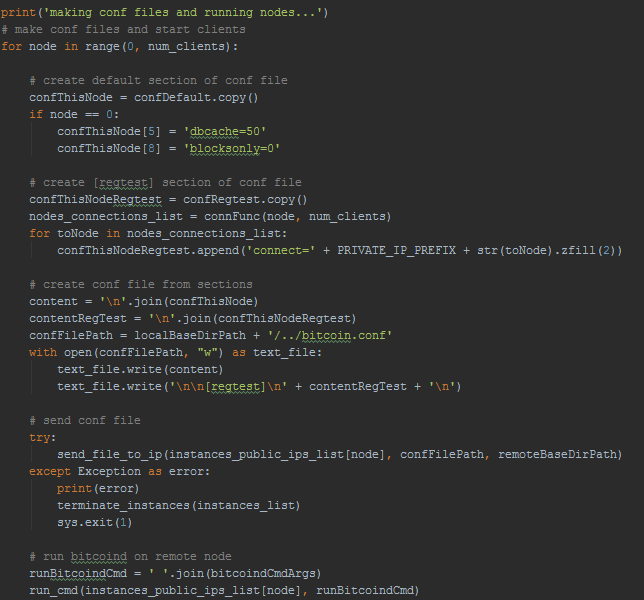
Copying data to instances

After creating the instances, the script copies to them the data directories it created before. The data directories are then unzipped on the instances and are ready for use. In addition, the two scripts “block” and “server” are also copied to the instances (more on those scripts later on).



Creating configuration file

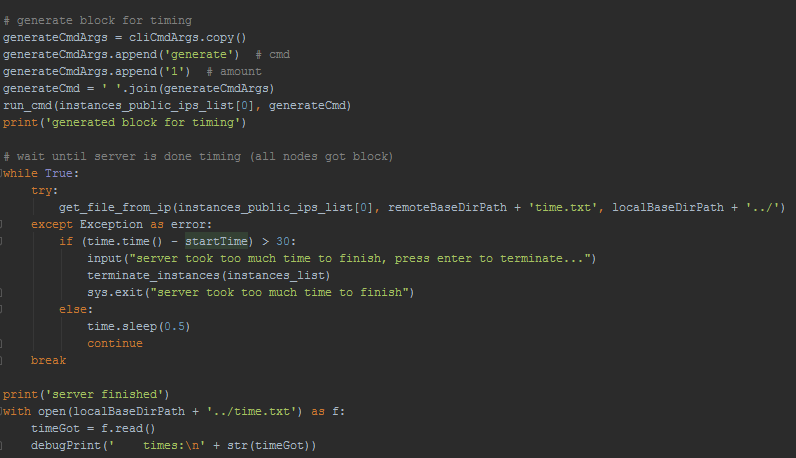
Now, after the instances are set up with all the data they need, we need to create a “.conf” file that will be used when running the bitcoin core client. The “.conf” file contains certain parameters that the bitcoin server will use when starting up, some of which are pre-determined by us as they should be the same for every run, while others are determined by the arguments passed to the script (as explained previously). The two exceptions are the “mempool” and “blocksonly” options, which are different for the node that mines the block and times the dissemination than it is for the other nodes. The reason for this, is that the mempool option determines if nodes save already validated transactions in order to speed up the validation process of incoming blocks, and we want them to validate blocks from scratch. With the miner we want to speed up the process as much as we can, so we want him to validate transactions as quickly as possible. The same goes for the blocksonly option which prevents the nodes for getting information about transactions and UTXOs from other nodes. After the “.conf” is created, it is copied to the AWS instances and we run the bitcoin core client.



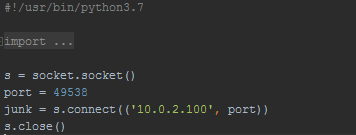
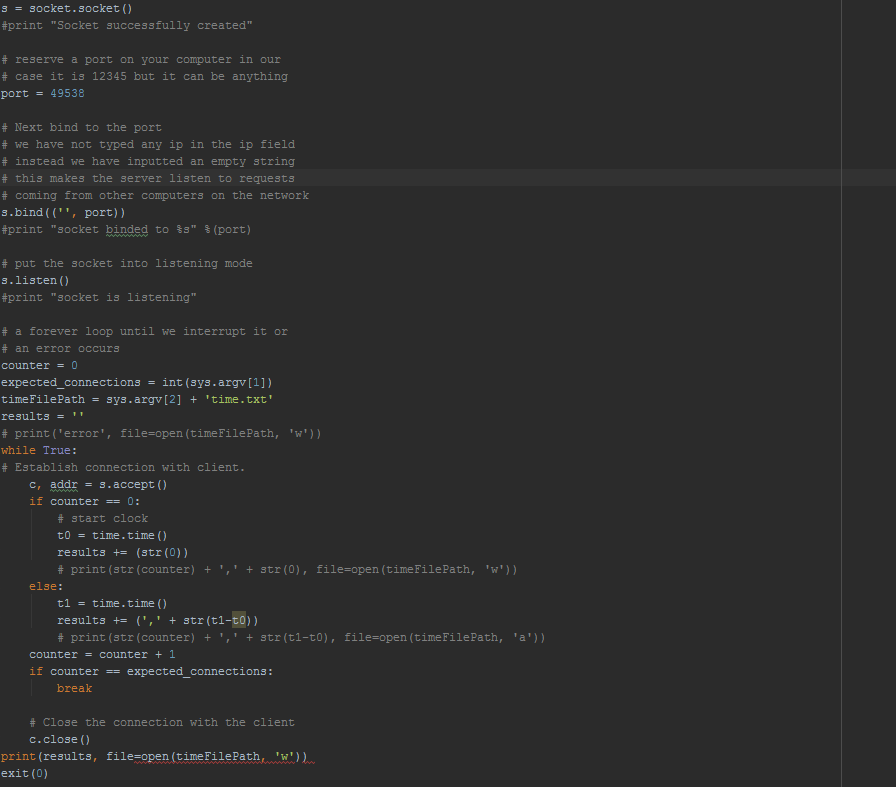
Mining and dissemination timing

In this final stage, tell our mining node to mine a new block, send it to its peers and measure the time it takes the block to reach all of them. This is done with the help of two scripts: “block.py” and “server.py”. The mining node starts timing and sends the block to all other nodes. The server script runs on the mining node and listens on a certain socket and waits. Meanwhile, when the other nodes receive the block, they run the block script, which attempts to connect to the miner node on the socket it opened. Upon opening the connections, the miner considers that node as having received the block. Upon receiving a connection from the final node, the miner stops timing and writes the result to a file.

Mining the block and timing its dissemination



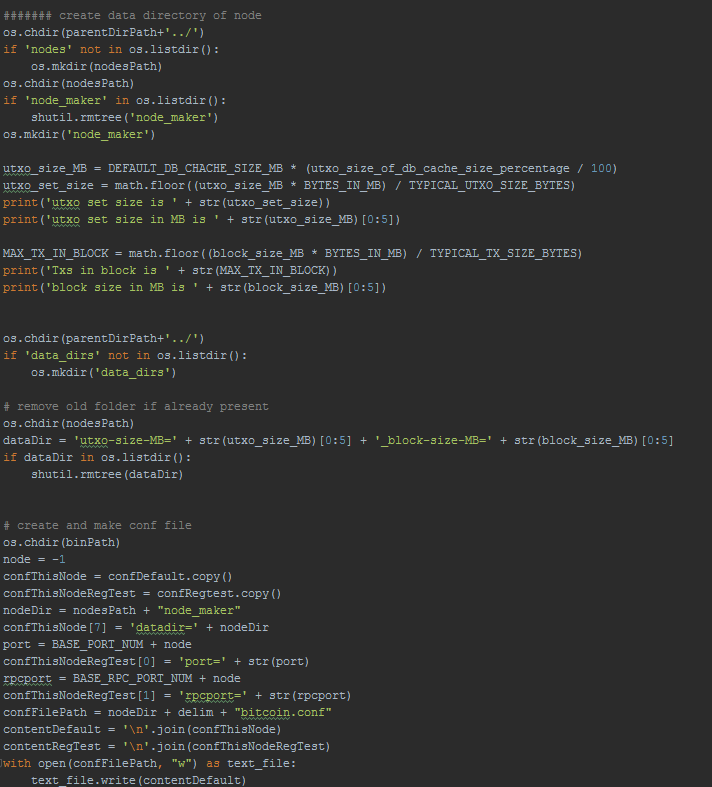
server.py



block.py

Script – create**\_**starting**\_**blockchain

This script is called by the make\_setup\_test script to create the starting blockchain for all the nodes. The running of this script is deterministic, so as to create the same starting point for every test that is run. The starting blockchain is constructed using the parameters of block size and UTXO set size that were passed to the main script. The blockchain is created locally and will be copied to the AWS instances later (as explained in the previous part about the main script). The starting blockchain is of course identical for each node. The process consists of creating transactions to validate and put into blocks, which are then inserted into the blockchain.



Script – run**\_**multiple**\_**tests

For convenience, another script is supplied, called run\_multiple\_tests.py, which can run multiple subsequent tests according to lists of parameters inside it. The script will go over every combination of parameters from all lists and can do multiple iterations for each combination (the parameters can be changed in the code). this script is intended only for parameter combinations that already have a corresponding data directory in the AMI, or at least will work for only one iteration (subsequent iterations will result in an error, because the main scrip will think the folder created on the first iteration is at the AMI too). Results file (generated by main script) is renamed to reflect the parameters used as a CSV file, and the content is appended to an aggregated results file.

