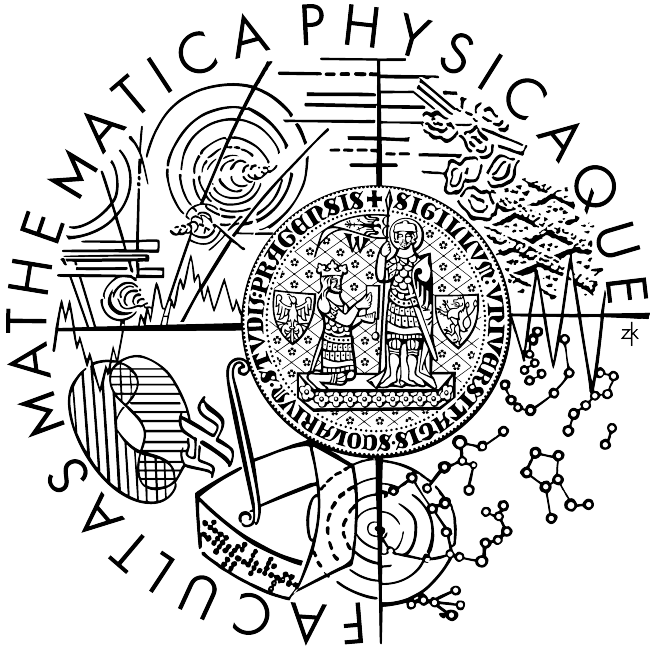
Charles University in Prague

Faculty of Mathematics and Physics

BACHELOR THESIS



Tomáš Filípek

Efficient Network Backup System

Department of Applied Mathematics (202. • 32-KAM)

Supervisor of the bachelor thesis: Mgr. Petr Baudiš

Study programme: Informatika

Specialization: Obecná informatika

Prague 2013

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In…..... date..... signature

Název práce: Efektivní síťový zálohovací systém

Autor: Tomáš Filípek

Katedra / Ústav: Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Petr Baudiš, Katedra aplikované matematiky

Abstrakt:

Pro ukládání vzájemně podobných souborů existují různé elegantní algoritmy, stejně jako pro jejich rychlé posílání po síti. Je však možné tyto algoritmy zkombinovat a vytvořit tak síťový zálohovací systém? S využitím architektury klient–server jsme takový systém sestavili. Mezi jeho základní vlastnosti patří podpora ukládání více verzí souboru a možnost inteligentního udržování historie. Abychom mohli určit vhodné hodnoty některých parametrů použitých algoritmů, provedli jsme měření výkonu aplikace. S použitím naměřených optimálních hodnot program funguje rozumně efektivně v čase i prostoru.

Klíčová slova: rsync, záloha, Myers, síť

Title: Efficient Network Backup System

Author: Tomáš Filípek

Department / *Institute*: Department of Applied Mathematics

Supervisor of the bachelor thesis: Mgr. Petr Baudiš, Department of Applied Mathematics

Abstract:

There are elegant algorithms for storing similar files, as well as for sending them efficiently through a network. But can these algorithms be combined to form a back-up system? We have constructed such a system using the client–server architecture. Its main features include support for versioning and intelligent history keeping. To assess the right values for some of the algorithm parameters, a simple performance testing was performed. Using the chosen values, the application is reasonably efficient both in space and time.

Keywords: rsync, backup, Myers, network

Contents

Introduction / Preface

The main reason why backup software exists, is to make sure you do not lose your data. Therefore, it would seem easy to say that its only important property is a high level of reliability. However, experience shows it is not a sufficient condition for making a system successful. The problem lies in making the software run efficiently both in space and time. Because, if every backup takes too much storage space, the user will think twice about doing the backup, which is bad. Backups should be done often and regularly, not only when it is absolutely necessary. The same applies to the case when the backup process takes too much time, which might happen when there is a slow connection between the client and server, for instance.

There are many decent solutions in the backup field available for free, some of which deal nicely with reducing the network traffic. This is usually done by employing the rsync algorithm. However, none of those makes use of the algorithm's internals to save disc space by finding similarities among files and storing the common sections only once.

The goal of this work is to provide a dependable backup system with a basic client – server architecture, capable of storing files paired with a reasonably long history. The fundamental aspect of the system is its efficiency both in space and time, employing techniques from rsync and diff algorithms. Because many decisions about various algorithm parameters have to be made, the program is tested and according to the results, optimal parameters are chosen. The testing methodology, results and conclusions form an integral part of the work.

The thesis begins with an introduction to the algorithms and methods used in the system. Because no existing implementation of them proved useful, they have been completely reimplemented to match the needs of the system. Every change from the original algorithms will be thoroughly described in this section.

Next section introduces the project structure and implementation. Since we have chosen the client-server architecture, the used protocol specification is also included. Finally, as a part of the implementation section, the serialization process is described. The reason is, the application is built on the Java platform, which is object-oriented and the default serialization technique is not always the most efficient solution.

In the following section a simple user guide is presented, which describes how to run all the basic parts of the application – the server, client, scheduler ...

The last section presents the performance testing process. It begins with a discussion about what we measure and why, and follows with an overview of the employed testing methods and testing data. Finally, the testing results are discussed and it is explained what conclusions have been drawn from the results.

Related Works

rdiff-backup – A backup tool capable of copying an existing directory into another, possibly a remote one. It uses the rsync algorithm to reduce the network traffic. For each file, a simple history is maintained, which is achieved through usage of difference scripts. However, the software is not fully optimized for minimizing disc usage on the server side. There are no shared blocks of data among similar files. Also, the way difference scripts are used implies that only similarities among adjacent versions can be capitalized on. For example, if we change a file periodically in a way that all the odd and all the even versions are similar, no data redundancy is detected. Another minor issue is that rdiff-backup is not completely portable, since the Windows version has not gone through its testing phase yet.

rsync – The original program which introduced the rsync algorithm. It was intended to supersede *rcp*, so the main and only advantage over the prior is its bandwidth-efficient algorithm. No history is kept of the transferred files. Also, no solution for reducing the disc space usage is provided.

GNU diff – A classic utility used for comparing two text files and creating a difference script. Since 1980s, it employs Myers' algorithm, which is also used in this work. However, only text files are supported.

Duplicati – A complex back-up software suite, actually a .NET reimplementation of a different program Duplicity, which was limited to the Unix platform. It provides users with many useful features such as data encryption or support for cloud systems, but the core functionality is the same as with rdiff-backup. Thus, there is no extra emphasis on reducing disc space usage on the server side.

1. Overview of the employed algorithms

1.1 Rsync

Originally described by Andrew Tridgell in his thesis[1], rsync algorithm deals with efficient synchronization of files over a slow network. Its main advantage over a plain file copy program like *rcp* is that it sends just the parts of the file that have changed, taking advantage of an existing older copy of the file.

The process of coping a file from Sender to Recipient roughly consists of the following steps:

- If there is an existing older version of the file at the recipient machine, it parses the file into equally sized, non-overlapping blocks. Otherwise, a simple copy is carried off.
- For each block of data, a weak and strong hash values are computed and the sets of all the weak hash and strong hash values are sent to the Sender, along with the chosen block size N.
- Sender reads the input file with the "window" method. That means at one time, only N adjacent bytes are looked at, starting with the section from position 0 (counting from 0) of length N. The window is gradually moved by one byte at a time to the next position, meaning the first byte is removed from the window and the first byte to the right of the window is added. The process ends when the right side of the window has reached the end of the file.
- For each position of the window, a weak hash value is computed. If it is found in the set received from the Server, the strong hash value is computed and searched for in the second set. If a match is found, it is assumed that the contents of the window are the same as that of a block on the Server. The strong hash function is assumed to be safe enough not to treat possible hash collisions.
- If the aforementioned match is encountered, all the previous unmatched data is sent to the Server. Then a pair of hash values identifying the current window contents is sent instead of the actual data. This step is essential to saving network bandwidth.
- The Server reconstructs the file and saves it.

Of course, for the algorithm to work reasonably fast, a special weak hash function must be utilized. In the original rsync, Adler-32 checksum has been used, in the form as described in an RFC document [6]. Because it is computed very frequently – at almost every position of the input file, it is beneficial that the value can be obtained just from the following (k – position of the current window, N – length of the window):

- $Adler32(k-1)$
- $data[k-1]$
- $data[k+N-1]$

Therefore, it is not necessary to go through all the bytes in the block when computing weak hash value of the window moving to a next position in the input file.

The Adler-32 checksum is computed by keeping two auxiliary values A and B. Let

$$n = 65521$$

$$A = \sum_{i=k}^{k+N-1} data[i] \pmod n$$

$$B = \sum_{i=k}^{k+N-1} (k+N-i) * data[i] \pmod n$$

Note that these values can be easily updated when moving the window by one byte:

$$A_{k+1} = A_k + data[k+N] - data[k]$$

$$B_{k+1} = B_k - (N * data[k]) + A_{k+1}$$

The actual checksum is then obtained by:

$$Adler32(k) = A_k + (p * B_k)$$

where $p = 2^{16}$

The rsync algorithm, as described in Tridgell[1], uses the MD4 algorithm to compute the strong hash. In later versions, the rsync utility switched to the MD5 algorithm.

Note: Our implementation slightly differs from the original rsync. Differences:

- n is set to $n = 2^{31} - 1$
- p is set to $p = 2^{32}$
- The SHA-256 hash function is used as the strong hash.
- Only weak hash values are downloaded to the Sender. Strong hash values are then compared online, through a special type of request to the Receiver.

1.2 Myers' Diff

1.2.1 Problem Description

If we need to store many files with similar content, it is often a waste of space to save them in a plain form. The common sections are then stored individually for each of the files. However, if we are able to keep just the differences among the files, we can save a considerable amount of disc space. The question stands – is it possible to compute the difference between two files efficiently? A more formal definition of the problem follows.

Problem: Let A and B be two files. Find a shortest edit script, i.e. at least one of the shortest sequences of character deletions from A, character additions from B and no-operations, that will transform file A into file B.

1.2.2 Definitions and Used Terms

Eugene Myers [2] solved the above described problem using dynamic programming. His solution uses a few terms and definitions:

- *File A and B* – finite sequences of characters
- *N (M, resp.)* - the length of file A (B, resp.)
- *Operation* – an addition of a character from B into A, a deletion of a character from A or an empty operation doing nothing.
- *Shortest edit script* – a shortest such sequence of operations that transforms A into B. As there can be multiple shortest edit scripts, this denotes any one of them.
- *Edit graph* – as illustrated in figure 1, it is a lattice graph of size (M+1)*(N+1) with characters from A being paired with x-axis units and characters from B paired with y-axis units.
- *k-th diagonal* – a line in an edit graph defined as $y = x - k$
- *Length of a path* – the number of deletions and insertions along the path through an edit graph
- *Snake* – a single deletion or insertion followed by zero or more diagonal moves. Thus, we have another definition of the path length: the number of snakes on the path.
- *Distance of [x,y]* – a minimal number of snakes required to move from [0,0] to [x,y] in the edit graph.

1.2.3 Algorithm Description

The goal of the algorithm is to find a shortest path through the edit graph from [0,0] to [N,M], where only the following moves from point [x,y] are allowed:

- to [x+1,y] (a deletion from file A)
- to [x,y+1] (an insertion into A from B)
- if A[x] is equal to B[y], then a move to [x+1,y+1] is allowed (a diagonal move)

A basic form of the algorithm:

```
V := empty list of vectors
For each distance d, starting from 0, step 1:
      For each diagonal g,  from -d to d, step 2:
            find the longest reaching path LP from [0,0] , on the diagonal g
      Add a vector of the last snakes of the longest reaching paths to V.
      If the longest reaching path on any diagonal crosses the point [N,M], break
Construct the path from [0,0] to [N,M] by moving backwards through V and looking
for the previous snakes that end in the start point of the current snake.
```

Example:

File A : ABCABBA

File B: CBABAC

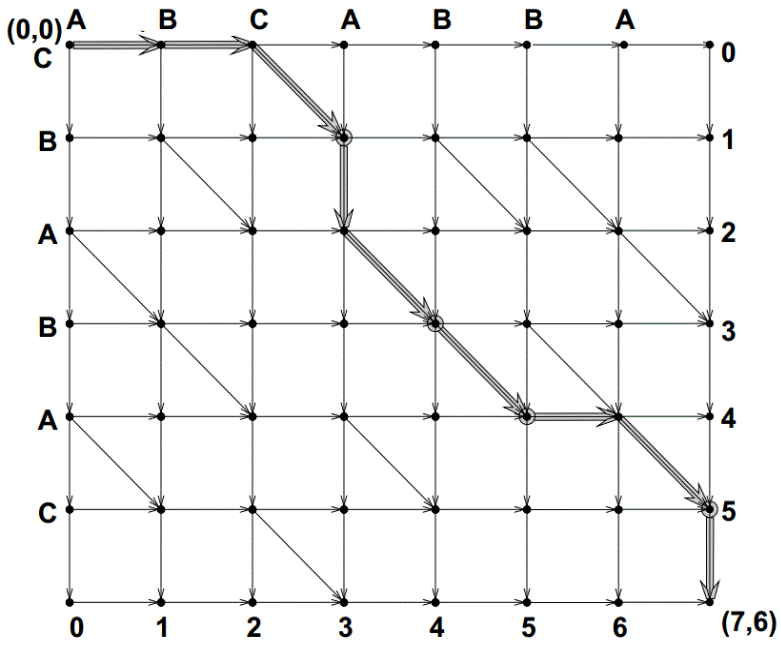


Figure 1 (source: E.Myers [2])

How to find the longest reaching path on diagonal k and on distance d ? Because a move along a snake changes the current diagonal by one, the last snake on the path must start at either $(k-1)$ -diagonal or $(k+1)$ -diagonal. Thus we can use the knowledge of the longest reaching paths on those diagonals at distance $d-1$.

Note: because at the odd (even) distances we can only reach to odd (even) diagonals, we can copy the furthest reaching snakes at odd (even) diagonals at an even (odd) distance from the vector at the previous distance. This allows us to take steps by 2 in the inner loop.

There is a drawback to this algorithm, though. Given length D of the shortest edit script, the algorithm needs to store D vectors of snakes, each containing at most $N+M$ members. That gives us a space complexity of $O((N+M)*D)$. For a rigorous analysis, see Myers [2]. A nice graphical depiction of the edit graphs used in the algorithm is available online at CodeProject [7].

1.3 A Linear Space Refinement for Diff

Eugene Myers [2] concludes that the above described form of the algorithm is not useful for larger files and offers a modification of the algorithm that has a space complexity linear with the size of the input files.

The linear space modification uses a notion of a *middle snake*. Let's define a middle snake as the $\frac{n+1}{2}$ -th snake on the path describing the shortest edit script. If we can find a middle snake, it is possible to construct the shortest edit script recursively, by using the divide-and-conquer method:

```
Shortest edit script path through (0..N)×(0..M) :
        If (M == 0) or (N==0), solution is trivial
        M <-- middle snake (suppose it start at [a,b], ends at [c,d])
        X <-- shortest edit script path through (0..a)×(0..b)
        Y <-- shortest edit script path through (c..N)×(d..M)
        Return concat(X,M,Y).
```

It can be seen that this algorithm only requires $O(N+M)$ memory.

To find a middle snake, we can use a modified basic version of the Myers' algorithm. The principal differences are:

- We construct the longest reaching paths simultaneously both from [0,0] and [N,M] for each distance d .
- During the construction of the longest reaching paths, we only remember one vector of distances on the diagonals. This is important, because since each time the method is called, one snake of the final path is found, and so the method is called at most D times at any moment. This gives us intuitively a $O(N+M)$ space complexity. For full analysis, see Myers [2].

The algorithm:

```
Find a middle snake through (0..N)×(0..M) :
        L <-- empty vector
        R <-- empty vector
        For each distance d, starting from 0, step 1:
                For each diagonal g,  from -d to d, step 2:
                        find the longest reaching path LP from [0,0] , on the diagonal g
                        L[g] <-- last snake on the path LP
                For each diagonal g,  from -d to d, step 2:
                        find the longest reaching path RP from [N,M] , on the diagonal g
                        R[g] <-- last snake on the path RP
                For each diagonal g,  from -d to d, step 2:
                        if L[g] and R[g] cross at any point, return L[g] as the middle snake
```

1.4 A Heuristic for Myers' Diff

The diff algorithm, as described in section 1.3, is good for finding the shortest edit script. However, in practice it is often preferable to find an edit script that is not the shortest one, but can be computed much faster. In this work, a heuristic developed by Paul Eggert [8] is used, which guarantees that a middle snake is found within a constant distance from [0,0] in the edit graph. However, the returned middle snake may not conform to the definition, because it may either be another snake on the shortest path through the edit graph or it can even lie on a different (and longer) path.

The principal idea is that the search for the middle snake is conducted up to a certain distance from [0,0], and if the middle snake is not found until then, the search is stopped. For each diagonal, the longest reaching snakes are checked and the one that reaches the furthest from [0,0] gets returned as the middle snake, even though it may be a suboptimal result.

Here is the modified algorithm for finding a middle snake:

```
Lim <- a constant

Find a middle snake through (0..N)×(0..M) :
       L <-- empty vector
       R <-- empty vector
       For each distance d, starting from 0, step 1:
              For each diagonal g,  from -d to d, step 2:
                     find the longest reaching path LP from [0,0] , on the diagonal g
                     L[g] <-- last snake on the path LP
              For each diagonal g,  from -d to d, step 2:
                     find the longest reaching path RP from [N,M] , on the diagonal g
                     R[g] <-- last snake on the path RP
              For each diagonal g,  from -d to d, step 2:
                     if L[g] and R[g] cross at any point, return L[g] as the middle snake
              If d >= Lim then :
                     ls <- furthest reaching snake (measured from [0,0]) out of L
                     rs <- furthest reaching snake (measured from [N,M]) out of R
                     if (ls.end - [0,0]) >= ([N,M] - rs.end)
                            then return ls
                            else return rs
```

2. Description of the Implementation

2.1 Overview

2.1.1 Program Description

The application is divided into two parts – one is server-side, the other is on the client machine. The former is mainly used for handling the data storage, i.e. dealing with blocks, log files, versions of files and composition and decomposition of files into/from blocks of data. It is operated through a network connection by the client part, which is, in contrast, more user-centric. Its main responsibilities include inspecting the newly inserted files, determining the parts of files to be sent and, importantly, communicating with the user.

The basic operations available at the client side:

- add a new file or directory into the system
- add a new version of an already present file
- obtain a file, either the latest or an older version
- delete a version of a file
- list all the contents

The server is able to store multiple versions of a file. Thus, it is essential to use an intelligent way of storing the file data, because various versions of a file will probably have some common data sections. Two approaches to storing file data are considered:

- blocks of data
- edit script

If the block-form is used, file data are parsed into blocks of a constant size, that is used globally for all files. These blocks and their hash values are useful when adding new files to the system - the incorporated rsync algorithm uses them to find matching sections in new files. The contents of each block are stored as a separate file. The process of determining the block size is described in a separate section.

If the script-form is used, the new version of the file is stored as an edit script relative to an existing older version of the file. The classic Myers' algorithm is implemented to compute the differences, along with a simple heuristic. However, the script-form is used only if the difference is small, otherwise the block-form seems more useful.

It is possible so specify a limit on the server database size. That means, if an addition of a new file should make the database size grow over the limit, the addition is rejected by the server and the client is informed. However, before that happens, the server tries to do a limited clean-up, which deletes all the block-form versions that are not the latest ones, including all the dependent script-form version. Since data blocks can be shared among all the present files and versions, it is necessary to monitor the reference count for each block to allow for deletion of unused blocks. The aforementioned cleanup first decrements the appropriate reference count for some blocks, and only after this phase is finished proceeds to the actual deletion of nowhere referenced blocks.

The server is able to serve multiple clients in parallel, where each client is served by a single thread. However, because of the character of the application, the space for parallelization is limited – critical sections including access to the file database must always run in exclusive mode.

The client part of the application consists of a functional core, bundled with a simple command line interface and a more user-friendly graphical interface.

On the client side, there is also a scheduler, which posts a set of previously defined requests to the server in fixed time intervals.

2.1.2 Example Behaviour

Case study 1:

Situation: The system contains a file F, with a single version V_1 of size around 5 MB. A new version V_2 of about the same size is added, with completely different contents.

Actions taken: Supposed the V_2 version does not contain any long common sections with version V_1, the whole contents of V_2 are sent to the server as plain data. Then, the creation of an edit script begins. Because both versions are around 5 MB of size and contain different data, the creation shortly stops – the script size grows over the script size limit, which is set to be equal to the block size. As a result, the version is stored in the block-form.

Case study 2:

Situation: The system contains a file F, with a single version V_1 of size around 1kB. A new version V_2, of size 5 MB, is added. The V_2 version was created as a concatenation of some new data and V_1 contents (in this order).

Actions taken: At first, the section containing the new data is sent to the server. Then, for the section identical to V_1, only the hash values identifying the data are sent. The server then tries to construct an edit script, but its size eventually grows over the script size limit, so the block-form is used instead, to store the version V_2.

Case study 3:

Situation: The system contains a file F, with a single version V_1 of arbitrary size. V_1 contains plain text. A new version V_2 is added. Version V_2 only differs from V_1 by fixing two typos.

Actions taken: The majority of V_2 contents matchea to existing blocks, so only the hash values are sent. Only those block that were affected by the typo correction are sent in its entirety. On the server, an attempt to create an edit script is made. Since the difference between V_1 and V_2 is small (typo corrections), the script is successfully created and the version is stored in the script-form.

2.2 Format of the Serialized Data Structures

The software is intended to be fully platform independent, so the format for data interchange over a network does not preserve any Java-specific or platform-specific features.
Variables of primitive types are serialized the following way:

- *byte*: byte variable is simply represented by its value
- *short, int, long* : each type is represented as consecutive bytes in big-endian order - *short* as two consecutive bytes, like as if the *short* has been cut into two bytes where the most important bits are in the first byte. *int* as four bytes and similarly, *long* as eight bytes.
- *float, double*: float variable is at first converted to the IEEE 754 format. Then, this *float* is looked upon as an *int* value containing the exactly same bits. The *int* variable is then used for serialization. d*ouble* is similarly converted to a *long* variable.
- *char*: variables of *char* type are handled the same way as *short* variables
- *boolean*: *true* is represented the same way as a *byte* variable of value 1, *false* corresponds to zero *byte.*

There is also an alternative way to serialize *int*s and *long*s. If it is used anywhere, it is explicitly noted so. This serialization is optimized for small values stored in the variable. The procedure is:

- The int/long variable is parsed into 7-bit blocks, starting at the least-important bits side.
- A bit (set to 1) is added to the left (most-important) side of each block, thus creating a byte from each block. The exception is the byte containing the most important bits of the original variable, where a zero bit is set, thus marking it is the final block.
- The bytes are then used in the little endian order, i.e. the byte containing the least important bits is sent first. This allows for sending just the non-zero blocks of the variable.

Enum variables are represented by their ordinal value, which is of *int* type. The actual ordinal values for the Requests class are mentioned later in this section.

The description of how reference types are handled follows.
String variables are handled in a special way. These steps are followed:

- String length (int value) is at the beginning. Each character of the String is represented as a series of bytes, where each byte is of a special format: the first one has special values at the two leftmost bits – the left bit signalizes that the string is encoded in UTF8, the right signalizes that another byte follows. The other six bits hold the actual data (the rightmost six bits of the int value). Next bytes are of this format: leftmost bit signalizes that another byte follows, the other seven bits hold the actual data (bits 12+k*7 to 6+k*7, where k=number of byte in the sequence minus two).
- The actual String value is sent as chars, one by one. Each char is sent as consecutive bytes: if the char is smaller than 0x007F, just one byte is sent. If the char is greater than 0x007F but smaller than 0x07FF, the sent bytes equal (0xC0 | char >> 6 & 0x1F) , (0x80 | char & 0x3F) . Else, if the char is greater than 0x07FF, three bytes are sent: (0xE0 | char >> 12 & 0x0F) , (0x80 | char >> 6 & 0x3F) , (0x80 | char & 0x3F) .
- If the String is known to be in ASCII format, a more efficient method is used. Each char is serialized as a byte, with the leftmost bit set to 0. The end of String is signalized as an extra byte at the end, with the value 0x80 .

Array variables are serialized as the following items in succession:

- an “alternatively” serialized integer marking the size of the array
- all the items in the array in their serialized forms (ordered the same way as in the source array)

Other reference types are never serialized in this application, but there is an exception – the Database class and all the supporting classes, which are handled is a special way. The serialization format of those classes follows – the items always come in succession, with primitives being serialized the usual way.

DBlock:

- int (the position in the hash collision list)
- int (reference count)
- int (block size)
- int (number of valid/used bytes)
- long (weak hash)
- String (strong hash)

DDirectory:

- String (name)
- int (number of items contained in the directory)
- for every item contained in this directory:
  - String (name)
  - boolean (is it a directory?)
  - the serialized DFile/DDirectory object

DFile:

- int (number of versions the file contains)
- for each version: the serialized DVersion object
- int (number of items on the path to this file)
- for each item on the path to this file: String (the name of the item)

DVersion:

- long (date&time as milliseconds since 01-JAN-1970, 00-00-00)
- String (file name)
- int (block size)
- String (strong hash of the version contents)
- long (size of the version contents, in bytes)
- boolean (is it in a script form?)
- int (number of blocks)
- for each block: the serialized DBlock object

Database:

- int (the number of present blocks)
- for each block: the serialized DBlock object
- int (the number of distinct weak hash values used)
- all the weak hash values that are used – each as a long
- int (the number of distinct strong hash values used)
- all the strong hash values that are used – each as a String
- int (total number of regular files in the database)
- for each file: the serialized DFile object

- int (number of items in the root directory)
- for each item:
  - String (name)
  - boolean (is it a directory?)
  - the serialized item (DFile/DDirectory)

The *Requests* enum class has the following ordering of its members:
0 = ITEM_EXISTS, 1 = CREAT_FILE, 2 = CREAT_VERS, 3 = CHECK_CHANGES, 4 = CREAT_DIR, 5 = DEL_VERS, 6 = GET_FILE, 7 = GET_ZIP, 8 = END, 9 = GC, 10 = GET_D_ITEM, 11 = GET_FS, 12=GET_SERVER_BLOCKS

2.3 Server

The core functionality of the server part of the application is contained in the Server class. When run, the program accepts connections from possibly multiple clients. Each client is served by a dedicated thread. For the specification of the communication protocol between client and server, see section 2.3. In the following, we will discuss all the important data structures used on the server.

Most of the administrative data is stored in an instance of the Database class, with only the edit scripts being kept in an extra structure. The reason is that unlike some of the Database members, scripts never need to be sent across the network.

Most important Database class members:

- blockSet – a set containing weak hash values for each present data block.

  Main usage: during the rsync algorithm “window” phase on the client side, to quickly rule out presence of the current block

- blockSet2 – a set containing strong hash values for each present data block

  Main usage: to prevent problems arising from hash collisions in the weak hash

- regularFiles – a collection of all data structures representing regular files (not directories)

  Main usage: makes it fast to go through all the versions of all the files when determining what to delete, in case of disc space shortage

- blockMap – maps names of the files containing the block data to the corresponding objects that represent the blocks

  Main usage: speeds up searching for a block object, for example when receiving new version of a file that has not changed much

- fileMap – represents the file system structure of the files committed to the program. Contains top-level files and directories.

  Main usage: preserving a hierarchical file system of the committed files

The fileMap structure is worth further discussion. Generally, it simulates a simple file system, where only files and directories are allowed (no special files like a pipe or symlink). As the client uploads a file to the server, it gets added to this filesystem.

The directory nodes are implemented using the DDirectory class, which contains the directory name and a structure holding the directory contents, just as fileMap holds the root directory contents.

Regular files are represented by DFile class, which contains its name, path to the file, and a list of all the file versions.

File versions are implemented using DVersion class. It includes information about its size, date of creation, strong hash of the contents, name of the file, whether it is stored in the script-form, and finally, a list of blocks that contain the version data (if block-form is used).

Data blocks also have an abstract representation, which is provided by the DBlock class. It contains the weak and strong hash values of the block data, reference count, size of the window (as described in rsync section) when the block was created. Further data elements include the byte count, which can be different from the window size, because if the file is not exactly aligned to blocks, the remainders are saved as standalone blocks, albeit smaller. Finally, also the block's position in the weak hash collision list is also included.

There is also an important data structure included in the Server class:

- scripts – maps versions to the edit scripts that represent them.

  Main usage: for each version stored in the script-form, it contains the script, which is used to reconstruct the actual data contents of the version.

In the following, we will discuss in depth all the common tasks and how they are dealt with on the server.

- Adding a new file/version

At first, the client checks whether the file to be added is already present. That means finding the target path in fileMap. Then, if needed, a request to create a directory or file object is sent to the server. What follows is the request to create a new version of the file. It is first checked whether the file contents have changed since the last uploaded version. This is done by comparing the SHA-256 checksum values. If the checksums do not match, the process continues. A request to add a new version is sent to the server, followed by its size. If there is not enough disc space, the server tries to do some clean-up by deleting some less important older versions of big files (see next section). Next, client sends some identification data about the version it is about to upload. The window loop, as described in the rsync section, begins. After the complete version data is obtained, it is intelligently determined whether to save the version in blocks or as an edit script – for details see following sections.

- Clean-up of the server database

First, a list of all versions stored in the block form is constructed. Those that are the latest block-form versions of the corresponding file are left out. Then the version objects are deleted, along with all the script-form versions depending on it. For each block that any of the deleted versions contained, its reference count is decremented. Finally, a simple garbage collection is executed over the present blocks, to prevent storing unused blocks in the server storage.

- Determining how to store a version (edit script / blocks)

First it is checked whether there exists a version of the specified file, that is stored in the block form. If not, block form is chosen. It is then determined whether the new version contains any newly parsed blocks, or just some of the previously present blocks. If there has been a new block parsed, it is assumed that script form is preferred and the script creation begins. However, if the script size grows over a limit, which is defined equal to the block size, the computations terminates and block form is used instead. The edit script is always relative to the last block-form version.

The diff-ing mechanism uses basically the same algorithm as *GNU diff*, but contains a few customizations. First, it works exclusively with binary data. The elementary units, upon which the

differences are calculated, are bytes. Another major modification is, it is possible to monitor the process of building the difference structure so that if the edit script grows over a specified limit, it can stop the construction and return appropriate status message.

- Obtaining a version of a file

At first a get request is received from a client. Then a copy of the DItem object is sent to the client, to help it decide about existence and type of the file to download, as well as about version numbers. If the *get zip* request was posted, the server delivers all the data as a single zip archive, otherwise a simple binary form is chosen. If the user wants to download a directory with all its contents, the client side of the application divides the request into multiple single file *get* requests. As a result, after receiving *get / get zip* command, the server sends contents of exactly one file.

- Zipping a file/directory

If the client requests a file/directory to be delivered as a ZIP archive, zipping process takes place after the relevant data are gathered. The built-in Java library implementation is used, and the archive is created in a ByteArrayOutputStream. Finally, the contents of the underlying byte array in the stream are sent to the client.

- Deleting a version of a file

After receiving version identification from the client, it is checked whether we are not trying to delete the last remaining version of the file. If that is the case, deletion is aborted. Otherwise, deletion proceeds. If there are any script-form versions that depend on this one, they are all converted to be relative to the adjacent version.

- Listing the contents of the server file database

Server sends the database file system root element object to the client. Further processing is done locally.

All of the server data, both administrative and file contents, are stored in a single directory, called “home directory” in the following sections. The server data structures are saved to disc each time a client disconnects. For serialization process specification, see section 2.2. There are two files for storing administrative data structures of the application. The first file, named “index”, includes an instance of the Database class. The second file is named “scripts” and contains the “scripts” data structure, as described above.

The contents of a data block are saved in a separate file. Its name consists of a hexadecimal representation of the block's weak hash value. However, if there are multiple different blocks with identical weak hash values, a suffix is added to the file name. Its form is “vXX”, where XX is a natural number denoting the position in the weak hash collision list. However, as blocks can be deleted, the hash collision list can have “holes” in it – once the position in the hash collision list is determined, the number never changes.

To send anything over the network to a client, the application uses an external serialization framework named Kryo. The main reason for choosing Kryo over the default Java serialization functionality was performance – Kryo framework is very minimalistic, and if we provide extra serialization procedure for each custom (not from Java library) class, it does no extra work than specified in the serialization procedures. The exact specification of how the custom classes are serialized, see section 2.2.

2.4 Communication Protocol

The communication between a client and the server consists of multiple independent requests, that are sent to the server. Each request begins with an instance of Requests type, and according to its value, further steps differ. In the following section, we will discuss these steps for each possible value of Requests type.

If an object instance is sent, its serialized form is used, as described in section 2.2.

Example:

Server → Client : long ... The server sends a long variable to the client.

ITEM_EXISTS

- Client → Server : String array (a path in the server database)
- Server → Client : boolean (whether the specified path points to an existing item)

DEL_VERS

- Client → Server : String array (a path in the server database)
- Client → Server : int (version number)
- Server → Client : boolean (whether the deletion succeeded)

CREAT_FILE

- Client → Server : long (size of the file to be added)
- Client → Server : boolean (whether to check the available disc space)
- Client → Server : String array (a target path in the server database)
- If the available space is being checked:
  - Server → Client : boolean (whether the file creation succeeded)

CREAT_VERS

- Client → Server : boolean (whether to check the available disc space)
- Client → Server : long (size of the file to be added)
- Server → Client : int (block size used)
- If the available space is being checked:
  - Server → Client : boolean (whether it is possible to add the version)
    ..if negative, request handling ends.
- Client → Server : String array (a path to a file in the server database)
- Client → Server : String (strong hash of the file contents)
- A loop begins, with the client each time sending one of the following:
  - String “raw_data“ followed by a byte array containing new data
  - String “hash“ followed by:
    - long : the primary hash of an already present block
    - String : the secondary hash of the block
  - String “get_vals“ followed by:
    - Server → Client : number of all weak hash values in server database
    - Server → Client : all the weak hash values (as long variables)
  - String “check” followed by:
    - Client → Server : String (a strong hash)
    - Server → Client : boolean (the block specified by the strong hash exists)
  - String “end“ , which marks the end of communication.

GET_FILE

- Client → Server : String array (a path in the server database)
- Client → Server : int (version number)
- Server → Client : boolean (whether the specified version is present)
- If the version is present:
  - Server → Client : byte array (contents of the file)

END

- Nothing follows, server shuts down.

CREAT_DIR

- Client → Server : long (size of the directory contents)
- Client → Server : boolean (whether to check the available disc space)
- Client → Server : String array (path on the server to be created)
- If the available space is being checked:
  - Server → Client : boolean (whether the directory creation succeeded)

GET_ZIP

- Client → Server : String array (a path in the server database)
- Client → Server : int (version number)
- Server → Client : boolean (whether the specified version/directory is present)
- If the version/directory is present:
  - Server → Client : byte array (a ZIP archive)

CHECK_CHANGES

- Client → Server : String array (a path in the server database)
- Client → Server : String (a content hash)
- Server → Client : boolean (whether it is reasonable to upload a new version)

GC

- No communication follows, only garbage collection over the server data blocks is performed.

GET_D_ITEM

- Client → Server : String array (a path in the server database)
- Server → Client : byte (0 .. the desired file does not exist, 1 .. the file is a directory, 2 .. the file is a regular file)
- Server → Client : the serialized DItem object (if it exists)

GET_FS

- Server → Client : Map<String,DItem> (the “fileMap” object, serialized the same way as inside a Database object)

GET_SERVER_BLOCKS

- Server → Client : Map<String,DBlock> (the “blockMap” object, serialized the same way as inside a Database object)

2.5 Client

On the client side of the applications, the following functionalities are present:

- Command line client
- Client with GUI
- Scheduler

The functional core of the client is included in the Client class, together with a simple command line interface.

When run, the Client class checks for program parameters containing the identification of the server to connect to. If it is not found, the program interactively asks the user to enter the data.

User requests are then read from the standard input in a loop, in work() method. After some pre-processing in serveOperation(..), the requests are distributed to specialized methods in switchToOperation(..).

In the following, we will discuss how the user requests are dealt with on the client side:

- Adding a new file

Adding a new file or version is utilized by the serveAdd(..) method. First, it is checked whether the file to be added exists and is not a symbolic link – if it is, the process is aborted. Symbolic links are not allowed to store in the system, because they are not a platform-independent feature. Next, a request is sent to server to determine, whether the file to which we want to add the version to, is already present in the database. If it is not, it must be created first. Then, it is determined whether the file is a regular file or a directory. Directories are handled differently – the process of adding is run recursively on each file the directory contains. If we deal with a regular file, its contents are read into memory and a strong hash is computed on it. It is compared with the contents hash of the last version on server. If it has not changed, nothing more is done. Otherwise, a request to add a new version is sent to the server with some information about the version and the window loop, as described in rsync section, starts – it is implemented in the windowLoop(..) method.

- Obtaining a file from server

The process is implemented in the serveGet(..) method. It checks whether the requested file is present on the server, whether the local target is directory when downloading a directory, and does other similar preprocessing of the request. The actual transmission is handled by methods receiveVersion(..) and receiveDirectory(..). There, in receiveVersion(..), the target file is created (if a directory has been defined as the target). The raw data is retrieved from the server and written into the target file.

- Deleting a version from server

A request to delete the specified version is sent to the server.

- Obtaining the database contents

An instance of the database filesystem root element object is retrieved from the server and printed in a structured way to the standard output.

The graphical interface client functionality relies on the Client class, described above. The GUI is implemented using standard Java libraries Swing and AWT. There are two main frames used - “connectFrame” provides options to connect to a server, whilst “browserFrame” is opened only

after a connection is established. It gives the user a tree-like view of the server database filesystem, along with basic options to manipulate the files – add a file/version/directory, obtain file/version/directory plainly or as a ZIP archive.

The scheduler is operated from the command line. The following information is required to run the scheduler:

- Address of the server, including the used port
- Path to a text file which contains user requests, one per line
- Time interval between consecutive executions of the scheduled code
- Maximum number of executions of the scheduled code before the scheduler shuts down. If this parameter is set to 0, no limit is set on the number of executions.

In regular intervals, it connects to the server and executes the user commands found in the input file. It uses the Client class for communication with the server.

3. User Guide

3.1 Command-Line Client

The command-line client can be run by executing the following scripts, found in the bin directory:

- console_client.bat (on Windows)
- console_client.sh (on UNIX)

However, it is usually necessary to specify program parameters, such as the address of the server. To do this, open the aforementioned script in a text editor and edit some of the following variables:

- COMP – the address of the server
- PORT – port used by the server
- LOCALE – localization option, possible values: EN, CZ

The script is then in charge of correctly passing these values as program parameters to the Client class. After the applications starts, it starts to ask for user commands in a loop. The following user commands are supported:

- **add** <SOURCE_FILE> [TARGET_FILE]

Purpose: To add a new file, directory or a new version of an already present file to the server file database.

Notes: <SOURCE_FILE> denotes the name of the local file to be added, [TARGET_FILE] is the path on the server machine (default value is equal to the filename of the source file). If the target file is already present, a new version is added, if not, a whole new record is created.

- **get** <SOURCE_FILE> <DESTINATION> [SOURCE_VERSION_NO]

Purpose: To obtain a file, version or directory from the server.

Notes: <SOURCE_FILE> is the name of the file to get. [SOURCE_VERSION_NO] is an optional argument. If it is not specified, the current version is fetched. If its value is a negative integer n, (current-n)-th version is obtained, if the value is a non-negative integer n, n-th version is requested (counting from zero as the first version inserted). <DESTINATION> denotes a path on the local machine, to which the downloaded file(s) will be saved.

- **get_zip** <SOURCE_FILE> <DESTINATION> [SOURCE_VERSION_NO]

Purpose: To obtain a file, version or directory from the server as a ZIP archive.

Notes: usage is exactly the same as in case of the "get" command.

- **delete** <PATH> <VERSION_NO>

Purpose: To delete a version of a file.

Notes: Deletion is possible only if there are at least two versions of the file present. The reason is to make sure that at least a single version stays available. <PATH> denotes a file in the server database, <VERSION_NO> is the number of the version to be deleted.

- **list** [verbose]

Purpose: To view the contents of the server database.

Notes: If "verbose" is specified, more information is printed, including the usage of every data block.

- **exit**

Purpose: To quit the client program and terminate the connection to the server.

3.2 GUI Client

The client with graphical user interface can be run by executing the following scripts:

- gui_client.sh (on UNIX)
- gui_client.vbs (on Windows)

No program parameters have to be specified before running these scripts. The server address and port, as well as localization of the client, can be specified in the main window of the application.

After starting the application, user enters the computer URI and port number of the desired server into the designated fields. A new window appears, consisting of a tree-like directory structure of the server file database.

To download a file from server, follow these steps:

- Select the source file or directory to download.
- Click the “Get“ button.
- In the file selection dialog, select the destination file or directory. In case the source is a directory, destination can only be a directory.
- Wait until the operation completes.

To download a file or directory as a ZIP item:

- Same as in the case above, just use the “Get ZIP“ button.
- It is possible to download a directory into a file, since it is downloaded in the form of a ZIP file.

To upload a file to server, follow these steps:

- Select the destination in the server browsing window. If the destination is a regular file, you can only upload a regular file. It would become a new version of the destination file regardless of it's name.
- Click the “Add“ button.
- Select the source file or directory.
- Wait until the operation completes.

3.3 Scheduler

The scheduler can be run by executing the following scripts from the bin directory:

- scheduler.bat (on Windows)
- scheduler.sh (on UNIX)

However, before running the scripts, it is usually necessary to specify the scheduler parameters, such as the scheduled commands. To do this, open the script in a text editor and edit some of the following variables:

- COMP – the IP address of the server
- PORT – number of the port used by the server
- TIME – time interval (in seconds) between two executions of the scheduled code
- INPUT_FILE – path to a text file which contains user requests, one per line
- COUNT – the maximum number of executions of the scheduled code. Zero value means there is no limit set.

The file pointed to by the INPUT_FILE variable contains user requests, as specified in the section 3.1 – one per line.

3.4 Server

The server can be run by executing the following scripts from the bin directory:

- server.bat (on Windows)
- server.sh (on UNIX)

It is usually necessary to specify program parameters such as the used port number, before running the script. To do this, open the script in a text editor and edit some of the following variables:

- PORT – port number used for accepting connections from clients
- HOME_DIR – directory to which all of the server data will be saved
- RESERVED_SPACE – Total space (in bytes) reserved for the application. The total space the server occupies will never exceed this limit.

4. Measuring Runtime Properties

4.1 Reasons for Measuring

During designing of the application, decisions had to be made about the size of certain parameters, notably:

- In the window loop: the block size used for a file
- In the heuristic in the diff-ing algorithm: the maximal distance to which the search for the middle snake spreads

Without measuring, it is unclear what values to use, since in both cases “bigger” values as well as “smaller” values have both advantages and disadvantages. Lets see what happens when choosing different block sizes:

Rather small leads to:

- Smaller space requirements, which leads to the ability to store more data in case a constant data space is dedicated to the application.
- Possibly worse dependability, as block sharing is utilized more than in the case of bigger blocks. I.e., if a block gets damaged, more files can get corrupted.
- If there are too many blocks of data, a time penalty may show up when uploading a new version of a file. Before the window loop, a set of all used weak hash values is downloaded, which can take some time. Also, if the weak hash values set gets too large, hash collisions may become more frequent.

Conversely, rather big leads to:

- Worse chances of sharing blocks between different files, which leads to bigger space requirements.
- Because the upper limit on the size of edit script on the server is set to be equal to the block size, bigger block size often makes the script computation take longer.
- The number of blocks is smaller than in the case of small block sizes, which possibly leads to a faster addition of files to the database.

As for the heuristics parameter, rather small limit leads to faster computation, but it is much further from optimal result than if a bigger value was used. If we get an extremely suboptimal result, the edit script takes more space and it also takes longer to apply it on a file.

For users, an important aspect of the system's speed is the relation between the file size and the time taken to upload it to the server database. We have carried out tests to illustrate the basic relation. Finally, it is also essential to know how the size of the database contents affects the time it takes to upload a certain file. Again, we have carried out tests to illustrate the basic relation.

The last two of the measurements are discussed in the last section of this chapter.

4.2 Testing Methods

To measure the time and space requirements of the application when different block sizes are chosen, we prepared a set of testing data. An overview of the testing data is charted in the following table:

Name	Dir / File	Size	No. of files inside	Description
similar	dir	1,59 MB	15	In the first five files, every two files differ by at most 1 kB. In the next five files, every two files differ by at most 8 kB. In the last five files every two files differ by at most 60 kB.
[DBI026] dbapl	dir	3,42 MB	12	Contains a set of files used during preparation for a CS course.
[NMAI062] Algebra I	dir	1,43 MB	156	Contains a set of files used during preparation for a CS course.
big_files	dir	19 MB	2	Contains two text files, the first is a series of 0s, the latter is a series of 1s
documents	dir	2,03 MB	12	Includes a few documents and images.
small_files	dir	5,4 kB	1000	Many small files holding just a few bytes.
WebApplication5	dir	4,43 MB	31	A typical Java web application project.
blank	file	263 kB	-	270000 of zero bytes
blank2	file	264 kB	-	271000 of zero bytes
JavaApplication1.java	file	346 B	-	346 bytes long source file
JavaApplication2.java	file	346 B	-	a slightly changed version of JavaApplication1.java
lipsum.txt	file	1,03 MB	-	Text file with random Latin words.
lipsum2.txt	file	1,32 MB	-	Text file with random Latin words.

Table 1 - Contents of the testing data set

Various types of data are included in the testing data set. The “similar” directory contains similar files, that will be used to test the performance of the edit script algorithm. Files from the “small_files” directory test the application performance if the input data are scattered across a bigger number of files. However, we believe that the most common usage of the system will be to back-up directories containing usual text document, spreadsheets, images, or programmer's source files. As a result, we have included mostly files of these types in the testing data set.

In the following by “scenario” we mean a set of user operations executed on the client side of the application. Considering the possible scenarios when using the application, we have concluded that there is a factor that distinguishes the scenarios most – the level of versioning functionality utilization. With this in mind, we have designed three typical scenarios. The first one, employes no versioning, but includes adding many different types of input data, from a few big files to a large number of very small files. The next scenario adds a little use of versioning, where around a half of the added versions will likely be stored in the script form and the other will likely be scripted. The last scenario employs the versioning functionality heavily. Many versions are added to the same file, with the difference between the consecutive versions gradually becoming bigger.

A summary of the designed scenarios:

Scenario 1 includes the following operations:

No.	Operation
1	add big_files
2	add big_files/file0 big_files/file1
3	add big_files/file1 big_files/file0
4	add small_files
5	add "/[NMAI062] Algebra I"
6	add "/[DBI026] dbapl"
7	add "/WebApplication5"
8	add documents
9	get big_files/file0 ../data/fileA
10	get_zip big_files/file0 ../data/fileB
11	get small_files ../data
12	get "[NMAI062] Algebra I" ../data
13	delete big_files/file1 0

Table 2 - Scenario 1 operations

In addition, there is an extra pair operations executed before anything else:

- add "/JavaApplication1.java" japp
- add "/JavaApplication2.java" japp

However, it is not included in the results of the measuring, because its only purpose is to warm-up the JVM.

Scenario 2 includes the following operations:

No.	Operation
1	add "/lipsum.txt" lipsum
2	add "/lipsum2.txt" lipsum
3	add "/lipsum.txt" lipsum
4	add "/lipsum2.txt" lipsum
5	add "/blank" blank
6	add "/blank2" blank
7	add "/blank" blank
8	add "/blank2" blank
9	get lipsum ../data/lipsumA 0
10	get lipsum ../data/lipsumB 1
11	get lipsum ../data/lipsumC 2
12	get blank ../data/blankA 0
13	get blank ../data/blankB 1
14	get blank ../data/blankA 2
15	delete blank 1
16	delete blank 0
17	delete lipsum 2
18	delete lipsum 0

Table 3 - Scenario 2 operations

Again, there is an extra pair operations executed before anything else:

- add "/JavaApplication1.java" japp
- add "/JavaApplication2.java" japp

Scenario 3 includes the following operations:

No.	Operation
1	add /similar/file0 similar
2	add /similar/file1 similar
3	add /similar/file2 similar
4	add /similar/file3 similar
5	add /similar/file4 similar
6	add /similar/file5 similar
7	add /similar/file6 similar
8	add /similar/file7 similar
9	add /similar/file8 similar
10	add /similar/file9 similar
11	add /similar/file10 similar
12	add /similar/file11 similar
13	add /similar/file12 similar
14	add /similar/file13 similar
15	add /similar/file14 similar

Table 4 - Scenario 3 operations

Again, there is an extra pair operations executed before anything else:

- add "/JavaApplication1.java" japp
- add "/JavaApplication2.java" japp

We have decided to run the performance tests locally. If the connection from a client to the server is led through a network, the main bottleneck, compared to the local connection, is the initial phase of the addition process. The set of weak hash values of all the present blocks is sent to the client. According to the total size of the server database, the estimated size of the set is charted in table 5.

DB Size	Appr. Set Size
10 MB	1,2 kB
100 MB	12 kB
1 GB	125 kB
10 GB	1,2 MB
100 GB	12 MB

Table 5 - Estimated hash value set size

It should be noted that the set size is directly dependent on the used block size. In table 5, the block size is chosen the same as in section 4.3, i.e. 64 kB.

To record the elapsed time per user operation, a small helper class cz.filipekt.Benchmark is used. It only calls the appropriate Client methods and records the time data.

For each scenario, the actual user commands are contained in a script file, called “benchmarkXX.bat”, with XX substituted for the number of the scenario. The script contains calls to the Benchmark class, thus measuring the elapsed time, but also manages to start up and shut down the server.

Java VM usually runs all methods in an interpreted mode at first. Later, after some statistics are collected, which is usually after a fixed number of executions, the methods get compiled. This is not a very convenient practice for benchmarking, so we have followed a special procedure. The JVM is run with a non-standard switch -XX:CompileThreshold=1, which causes all methods to get

compiled after just one execution. A mock request is then handed to the server to ensure the methods are compiled and the JIT process will not skew the performance measuring tests.

We have chosen three distinct block sizes to be tested:

- 1024 B
- 8192 B
- 65536 B

To eliminate other interfering elements, the middle snake limit has been fixed at 1024 operations.

To determine the optimal middle snake limit, we have in turn fixed the block size at 65536 B and tested the following limits:

- 128 operations
- 1024 operations
- 8192 operations

As a result, we have tested 5 different configurations in total.

The machine used for testing: Intel Core2 Quad Q9300 (4x2,5GHz), 8GB DDR2 RAM

Operating system used: Microsoft Windows 8 Pro, 64-bit version

4.3 Discussion of the Testing Results

Complete results of the tests carried out can be found in the Attachments section in tables A to O. Summary of the testing results is charted in the following table.

Set No.	Block Size	Heuristics Limit	Total Size	Time Elapsed (ms)
1	1024 B	1024 ops	20,6 MB	83842
2	1024 B	1024 ops	1,72 MB	8770
3	1024 B	1024 ops	432 kB	2824
1	8192 B	1024 ops	13 MB	29438
2	8192 B	1024 ops	1,17 MB	3360
3	8192 B	1024 ops	237 kB	4656
1	65536 B	1024 ops	12,2 MB	27686
2	65536 B	1024 ops	1,41 MB	5873
3	65536 B	1024 ops	447 kB	3269
1	65536 B	128 ops	12,2 MB	27634
2	65536 B	128 ops	1,41 MB	2855
3	65536 B	128 ops	447 kB	1536
1	65536 B	8192 ops	12,2 MB	27730
2	65536 B	8192 ops	1,41 MB	33518
3	65536 B	8192 ops	448 kB	41220

Table 6 - Testing results

For each configuration mentioned in table 6, tests have been run with 50 iterations for each of the three scenarios. The average elapsed time is included in the last column of table 6. Total size, as charted in the fourth column of table 6, denotes the total size of the server database directory after executing the user commands the specific scenario. In scenario 3, an additional garbage collection

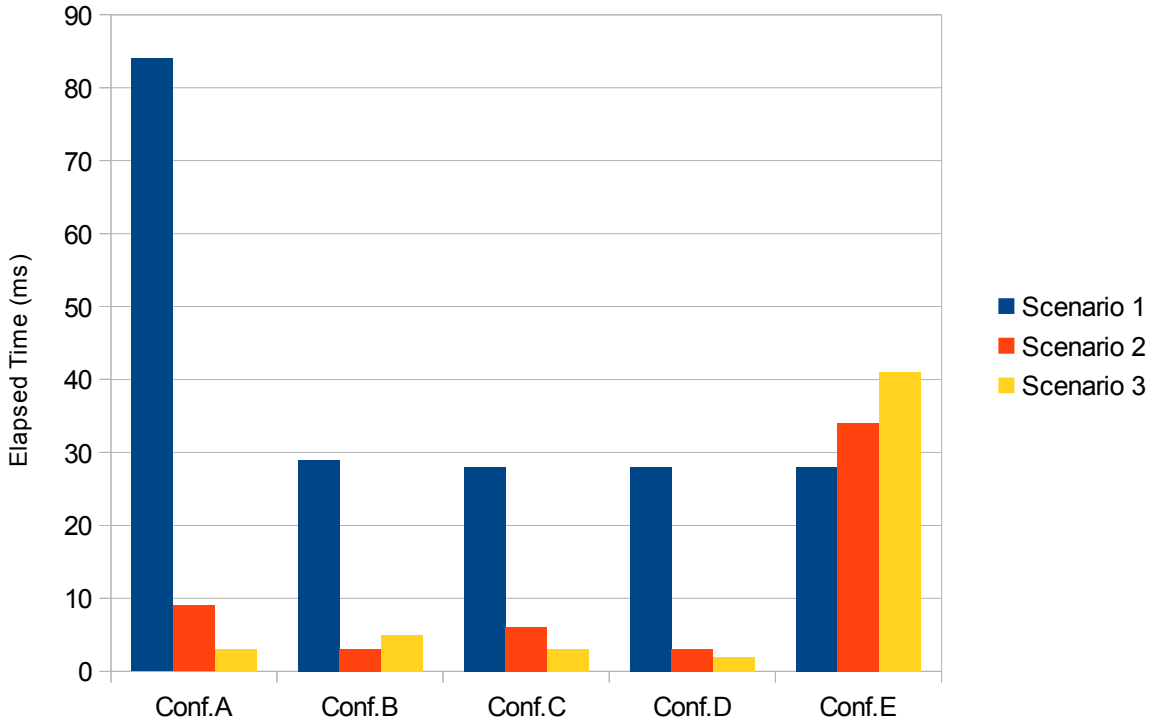
over the unused blocks was performed before measuring the database size, since this space would nonetheless be freed the first time it is needed.

As for scenario 1, the only impractical configuration is the first one, i.e. block size 1024 B and heuristics limit 1024 operations. On the other hand, all the configurations with block size equal to 64 kB seem to be quite equivalent, considering both time and space efficiency. This is understandable, because the only part of the application to be affected by varying the heuristics limit is the versioning mechanism. Scenario 1 does not utilize versioning, with the exception of the two warm-up operation at the beginning.

In scenario 2, the situation is different, since versioning functionality is now utilized. Right away we see that the last configuration, block size 65536 B and heuristics limit 8192 operations, is very slow, compared to the others. It is theoretically balanced by the fact that the created edit scripts are closer the optimum, since the heuristics limit is quite high. But compared to the smaller heuristics value configurations, the database size is almost not bigger at all, so there is no need to chose big heuristics limit, based on this scenario. The best-performing configurations are 8192 B block size, 1024 operations limit and 64 kB block size and 128 operations limit, based on this scenario. The slight rise in time complexity with a greater block size is presumably caused by the fact that the limit on the script size is set equal to the block size, thus the script computation takes more time before it gives up (or eventually succeeds).

The third scenario tests almost exclusively the versioning functionality. Similarly to the previous scenario, the configuration with a big heuristics limit is quite slow and is not much more economic in space, either. As expected, the fastest configuration is the one with the smallest heuristics limit value, i.e. 128 operations. However, the lowest space demands has, surprisingly, the configuration with 8 kB block size and 1024 heuristics limit. This is caused by the specific structure of the files that are added. The files begin with a series of 100000 zero bytes. As a result, the space efficiency gain in using scripts for bigger differences (8kB vs 64 kB) is outweighed by the effect of storing the initial monotonic series of bytes with a much smaller (single) block. Also, when a block-form is used, the smaller block size allows for greater amount of block sharing.

A graphical view of the measured time values follows in the graph 1. The configurations, as introduced in table 6, are named in order A,B,C,D,E,F.



Graph 1 - Results of the time performace measuring

It is interesting that in the first scenario the space demands actually decrease with the increasing block size. We believe that it is caused by the decreased time required to work with all the blocks, since there is a considerably smaller number of blocks present when a big block size is used.

We have decided to choose the optimal configuration as 64 kB block size and 128 heuristics limit (the D variant), because it was the fastest configuration. Although it was not the most space efficient one, the space demands remain quite low.

4.4 Performance from Users' Perspective

For purposes of the following tests, we have prepared a new set of test files, as charted in table 7.

Name	Size	Description
file_size_1MB	1 MB	Random bytes.
file_size_5MB	5 MB	Random bytes.
file_size_10MB	10 MB	Random bytes.
file_size_20MB	20 MB	Random bytes.
file_10M_XX	10 MB	XX substitues for 0 up to 19. Random bytes.

Table 7 - Test files

First, we measured the upload time of "file_size_5MB", depending on the server database size. For similar reasons as in section 4.2, the tests were performed locally, i.e. the network latency was not included. The series of operations, that were executed, begins with adding an adequate number of

10 MB files, to prepare the desired server database size. Then, the addition of the actually measured file follows, as charted in table 8.

No.	Operation
1	add file_10M_0
..	..
n	add file_10M_{n-1}
n+1	add file_size_5MB

Table 8 - Test Operations

Next, the upload time was measured, depending on the size of the file being uploaded. The initial database size was chosen to be 10 MB. The test scenario contained operations charted in table 9. Substitute {XX} with the file size in MB.

No.	Operation
1	add file_10M_0
2	add file_size_{XX}MB

Table 9 - Test Operations

All tests have been executed in 50 iterations.

The machine used for testing: Intel Core2 Quad Q9300 (4x2,5GHz), 8GB DDR2 RAM

Operating system used: Microsoft Windows 8 Pro, 64-bit version

The testing results (average values) are contained in tables 10 and 11. The first depicts the results of the first test, i.e. the relation between the upload time and the server database size.

DB Size (MB)	Elapsed Time (ms)
10	2028
50	2001
100	2001
200	2022

Table 10 - Test Results Summary

The latter charts the relation between upload time and the file size.

File Size (MB)	Elapsed Time (ms)	Time per MB (ms)
1	461	461
5	2028	406
10	3980	398
20	8042	402

Table 11 - Test Results Summary

As expected, the upload time is not much correlated with the server database size, if the server is run locally. This is ensured by using proper data structures with constant access times when possible. If it is run over a network, the initial transfer of weak hash values must be taken into account. As explained in section 4.2, its size is linear with the server database size.

The upload time seems to be linearly related to the size of the file being uploaded. A bitrate of ca. 2,5 MB/s was achieved.

For detailed results of the tests, see tables P and Q in the Attachments section.

Epilogue / Conclusion

The thesis covered the algorithms and principles behind the creation of an efficient network back-up system. An integral part of the work is the actual implementation, which is included as an attachment in the electronic form.

The first chapter introduced the rsync and diff algorithms, which are indispensable for the application to work efficiently in space and time. Since the original version of Myers' diff algorithm produces optimal results only with a non-trivial time complexity, a heuristic is also introduced in the chapter.

The second chapter described the implementation, starting with a high level overview of the system and following with a detailed account on each part of the system.

Next chapter provided the reader with a simple user guide, containing instructions on how to run the attached software.

In the final chapter we presented the testing process, including the reasoning about what to measure, the actual methodology description and the testing outcomes and discussion. We also provided the optimal values for the tested parameters, based on the testing results.

We believe there is a space for further work on making the application suitable for holding larger amounts of data. In the current state, the application works smoothly when up to 50-100 GB of data is kept, but the addition process slows down in a linear fashion with the database size.

Bibliography

[1] TRIDGELL, Andrew. Efficient algorithms for sorting and synchronization. Canberra 1999. PhD thesis. Australian national university. Thesis supervisors R.Brant, P.Mackerras, B.McKay.

[2] MYERS, Eugene W. An O(ND) difference algorithm and its variations. Tucson 1986. Department of Computer Science, University of Arizona.

[3] HUNT, Charlie and JOHN, Binu. Java Performance. 1st ed. Michigan: Addison-Wesley Professional, 2011. ISBN 978-0137142521.

[4] EVANS, Benjamin and VERBURG, Martijn. The Well-Grounded Java Developer. 1st ed. Manning: Shelter Island, 2012. ISBN 978-1617290060.

[5] KARP, Richard M. and RABIN, Michael O. Efficient randomized pattern-matching algorithms. Berkeley 1987. University of California, Berkeley.

[6] DEUTSCH, L. Peter and GAILLY, Jean-Loup. ZLIB Compressed Data Format Specification version 3.3. Request for Comments: 1950 [online]. 1996. Available at: http://tools.ietf.org/html/rfc1950 [cit. 2013-05-05].

[7] BUTLER, Nicolas. Investigating Myers' diff algorithm [online]. 2009 [cit. 2013-05-19]. Available at: http://www.codeproject.com/Articles/42279/Investigating-Myers-diff-algorithm-Part-1-of-2

[8] EGGERT, Paul. GNU Diff source code [online]. 1993 [cit. 2013-05-19]. Available at: http://www.opensource.apple.com/source/gnudiff/gnudiff-10/diffutils/analyze.c

List of Tables and Figures

Source Code

The source code for the associated piece of software can be obtained from two sources:

- the accompanying CD
- online, at GitHub

The attached CD contains a single directory, *save-the-world*, which is the name of the project. Build instructions can be found in the *instructions.txt* file located inside the directory.

The GitHub repository can be found at

https://github.com/filipekt/save-the-world [as of 18-05-2013]

Attachments

SET1 Blocksize 8192B Heuristics 1024 ops

Operation no. / Measurement no.	1	2	3	4	5	6	7	8	9	10	11	12	13	Total
1	6755	1716	1716	2184	1170	2059	2698	1170	843	7160	1170	484	13387	**29138** (ms)
2	6771	1732	1716	2168	1155	2075	2714	1201	827	7129	1154	499	13193	**29154** (ms)
3	6802	1748	1685	2215	1202	2090	2777	1201	827	7285	1155	499	12944	**29499** (ms)
4	6755	1747	1732	2356	1202	2106	2793	1201	858	7192	1124	499	13374	**29578** (ms)
5	6833	1748	1669	2340	1201	2106	2683	1248	858	7021	1061	514	12761	**29295** (ms)
6	6771	1732	1732	2200	1154	2075	2698	1170	843	7316	1155	484	12885	**29343** (ms)
7	6771	1732	1700	2293	1201	2075	2715	1217	842	7192	1155	499	12966	**29405** (ms)
8	6773	1747	1731	2137	1155	2075	2684	1170	827	7379	1138	483	13204	**29312** (ms)
9	6819	1716	1700	2121	1170	2075	2730	1170	842	7379	1061	499	12864	**29295** (ms)
10	6801	1669	1716	2340	1217	2106	2777	1217	843	7612	1076	514	12858	**29901** (ms)
11	6770	1716	1685	2340	1201	2091	2699	1216	842	7176	1186	499	12805	**29434** (ms)
12	6770	1685	1747	2309	1202	2153	2855	1264	858	7441	1154	499	12860	**29950** (ms)
13	6802	1716	1669	2340	1186	2075	2699	1216	842	7441	1029	515	12957	**29543** (ms)
14	6723	1763	1701	2308	1202	2044	2761	1233	843	7457	1092	499	12968	**29639** (ms)
15	6740	1747	1731	2325	1185	2059	2715	1201	827	7348	1139	500	12839	**29530** (ms)
16	6788	1732	1716	2309	1216	2091	2746	1201	842	7426	1123	499	12943	**29702** (ms)
17	6801	1732	1670	2340	1201	2106	2761	1217	842	7145	1123	500	12913	**29451** (ms)
18	6770	1732	1732	2106	1170	2059	2683	1185	842	7208	1139	468	13296	**29107** (ms)
19	6787	1779	1685	2387	1170	2090	2730	1217	827	7005	1030	484	12705	**29204** (ms)
20	6755	1732	1732	2262	1170	2075	2652	1170	827	7051	1077	499	12961	**29015** (ms)
21	6864	1732	1747	2246	1185	2060	2745	1201	858	7004	1124	515	12986	**29294** (ms)
22	6771	1716	1669	2340	1201	2075	2808	1216	843	7114	1076	499	12864	**29341** (ms)
23	6804	1732	1716	2324	1170	2075	2699	1202	843	7972	1077	499	12756	**30126** (ms)
24	6802	1732	1732	2309	1216	2059	2745	1248	842	7145	1154	515	13331	**29512** (ms)
25	6786	1731	1732	2293	1217	2106	2730	1201	842	7192	1124	500	13208	**29467** (ms)
26	6786	1747	1685	2356	1201	2122	2761	1185	827	7160	1155	499	12920	**29497** (ms)
27	6786	1747	1748	2106	1155	2106	2698	1185	827	7238	1201	483	12810	**29293** (ms)
28	6817	1747	1716	2137	1170	2075	2761	1185	842	7223	1014	484	12918	**29184** (ms)
29	6770	1747	1716	2340	1201	2107	2683	1185	842	7208	1076	499	12753	**29387** (ms)
30	6801	1732	1669	2387	1202	2106	2746	1216	842	8003	1123	515	12973	**30355** (ms)
31	6770	1700	1685	2262	1185	2059	2668	1170	842	7177	1014	500	12983	**29045** (ms)
32	6772	1731	1732	2325	1186	2090	2761	1201	826	7098	1061	484	12914	**29280** (ms)
33	6770	1747	1732	2308	1201	2059	2715	1217	827	7270	1124	499	13313	**29482** (ms)
34	6817	1731	1685	2308	1201	2075	2746	1201	827	7473	1030	515	12857	**29622** (ms)
35	6786	1732	1731	2137	1154	2075	2683	1186	827	7379	1154	499	12967	**29356** (ms)
36	6802	1732	1731	2309	1170	2106	2714	1217	842	7223	1139	514	12972	**29512** (ms)
37	6739	1763	1732	2153	1170	2044	2808	1201	827	7644	1139	483	12814	**29716** (ms)
38	6787	1747	1778	2293	1139	2075	2698	1186	827	7145	1124	499	13497	**29311** (ms)
39	6895	1731	1716	2090	1154	2060	2746	1170	811	6957	1138	484	13200	**28965** (ms)
40	6832	1747	1669	2277	1186	2044	2730	1185	843	7457	1045	484	12949	**29512** (ms)
41	6724	1670	1731	2262	1201	2106	2761	1217	842	7364	1154	499	12763	**29544** (ms)
42	6833	1732	1669	2340	1217	2200	2746	1248	827	7208	1139	499	12771	**29671** (ms)
43	6817	1731	1732	2278	1202	2075	2714	1202	827	7191	1061	499	13141	**29342** (ms)
44	6786	1747	1669	2294	1154	2090	2667	1185	827	7457	1014	499	12827	**29402** (ms)
45	6771	1747	1732	2137	1154	2044	2714	1155	827	7129	1155	484	13185	**29062** (ms)
46	6786	1794	1685	2371	1201	2106	2667	1185	842	7051	1154	515	12878	**29370** (ms)
47	6770	1732	1748	2293	1201	2075	2761	1217	827	7239	1092	499	13048	**29467** (ms)
48	6739	1716	1669	2325	1217	2090	2715	1233	842	7145	1139	500	12844	**29343** (ms)
49	6724	1731	1747	2324	1185	2090	2730	1201	827	7348	1139	499	12773	**29558** (ms)
50	6770	1731	1669	2309	1201	2121	2683	1201	843	7301	1030	499	12790	**29371** (ms)
AVG	**6785**	**1734**	**1711**	**2272**	**1187**	**2085**	**2727**	**1202**	**837**	**7278**	**1110**	**498**	**12974**	**29438** (ms)
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(µs)	

Table A

SET2 Blocksize 8192B Heuristics 1024 ops

Operation no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	Total
Measurement no.																			
1	593	1342	203	281	125	94	78	78	110	141	109	63	78	62	6555	3328	3176	18348	**3388** (ms)
2	577	1342	218	265	109	109	78	62	110	125	109	63	78	62	6568	3466	3314	18273	**3339** (ms)
3	561	1341	218	265	110	109	78	78	109	125	109	62	78	63	6510	3507	3233	18303	**3338** (ms)
4	577	1342	219	266	125	109	78	78	110	125	109	63	94	62	6717	3451	3346	18533	**3389** (ms)
5	577	1357	218	266	125	109	78	62	94	125	109	63	78	62	6559	3448	3176	18285	**3354** (ms)
6	577	1342	218	265	109	109	78	78	93	124	109	47	78	62	6551	3439	3167	18124	**3320** (ms)
7	577	1357	219	265	109	109	78	78	94	125	109	63	78	62	6500	3456	3196	18343	**3354** (ms)
8	593	1342	219	265	109	110	78	62	109	140	109	62	78	63	6435	3417	3190	18121	**3370** (ms)
9	578	1342	218	265	125	109	78	78	109	141	109	63	78	62	6634	3458	3222	18301	**3387** (ms)
10	562	1342	219	250	125	109	78	78	109	125	109	63	78	62	6553	3496	3171	18399	**3341** (ms)
11	577	1341	218	265	109	109	78	78	109	124	109	63	78	62	6368	3434	3237	18572	**3352** (ms)
12	578	1342	218	265	109	110	62	78	94	125	110	63	78	63	6769	3487	3186	18599	**3327** (ms)
13	562	1342	219	265	109	110	78	78	109	125	110	63	78	62	6454	3442	3202	18927	**3342** (ms)
14	593	1342	203	266	125	109	78	78	110	125	109	63	78	62	6496	3368	3303	18270	**3372** (ms)
15	577	1341	218	266	109	109	78	62	109	125	109	62	78	62	6717	3448	3193	18332	**3337** (ms)
16	578	1342	234	265	125	110	78	78	109	125	109	62	78	78	6551	3437	3250	18588	**3403** (ms)
17	578	1342	218	265	109	110	62	78	109	125	110	62	78	62	6397	3451	3218	18588	**3340** (ms)
18	577	1341	234	265	110	109	78	78	109	140	109	62	78	63	6575	3432	3302	18175	**3384** (ms)
19	578	1342	218	265	125	109	62	78	109	125	109	62	78	62	6511	3461	3231	18343	**3354** (ms)
20	561	1341	218	281	124	93	78	78	109	140	110	62	78	62	6396	3542	3221	18149	**3366** (ms)
21	578	1342	218	265	125	110	78	78	109	125	110	62	78	62	6495	3436	3258	18256	**3371** (ms)
22	592	1342	219	266	109	109	78	78	110	125	109	63	78	62	6431	3428	3200	18261	**3371** (ms)
23	577	1342	219	281	109	109	78	78	125	140	109	63	78	62	6430	3452	3272	18265	**3401** (ms)
24	577	1342	219	250	125	109	78	78	110	125	109	63	62	62	6431	3433	3187	18496	**3341** (ms)
25	577	1342	218	266	109	93	78	62	109	125	109	62	78	63	6552	3561	3204	18394	**3323** (ms)
26	562	1342	218	266	125	109	78	78	110	125	109	63	78	62	6635	3360	3166	18149	**3356** (ms)
27	578	1357	218	281	125	109	78	78	109	125	109	62	63	63	6501	3416	3229	18367	**3387** (ms)
28	577	1341	218	281	124	93	78	78	93	124	109	62	78	62	6436	3367	3248	18275	**3349** (ms)
29	561	1341	218	281	124	93	78	63	109	124	109	62	78	78	6409	3700	3370	18703	**3351** (ms)
30	577	1341	218	265	124	93	78	63	109	124	109	47	78	62	6481	3366	3211	18259	**3319** (ms)
31	577	1342	219	265	124	109	78	78	109	140	109	62	78	63	6391	3475	3243	18589	**3385** (ms)
32	578	1342	219	265	125	110	78	78	109	125	109	62	78	63	6535	3456	3179	18218	**3372** (ms)
33	577	1342	219	266	125	109	78	62	110	125	109	63	78	63	6400	3453	3187	18175	**3357** (ms)
34	578	1342	218	265	110	109	78	63	109	125	109	62	78	63	6458	3482	3184	18412	**3341** (ms)
35	578	1342	218	281	125	110	78	78	109	125	109	62	78	63	6706	3459	3228	18278	**3388** (ms)
36	577	1341	218	265	125	109	78	78	109	125	109	47	78	62	6461	3385	3195	18740	**3353** (ms)
37	562	1342	218	281	125	109	78	62	109	125	109	62	78	62	6482	3499	3246	18382	**3354** (ms)
38	562	1342	219	265	125	94	78	78	109	125	109	62	78	62	6532	3565	3182	18731	**3340** (ms)
39	593	1341	218	265	125	93	78	78	110	125	109	63	78	62	6460	3498	3245	18630	**3370** (ms)
40	577	1326	219	265	109	109	78	78	109	124	109	63	78	62	6533	3609	3228	18327	**3338** (ms)
41	577	1342	219	280	125	109	78	78	109	125	109	63	78	62	6422	3464	3210	18368	**3385** (ms)
42	592	1341	218	265	109	109	78	78	94	125	110	62	78	62	6560	3513	3173	18295	**3353** (ms)
43	577	1342	219	280	125	94	78	62	109	125	110	63	78	62	6572	3439	3290	19177	**3356** (ms)
44	577	1326	234	281	124	109	78	63	109	124	109	47	78	62	6536	3407	3518	18674	**3353** (ms)
45	577	1357	219	265	125	110	78	78	109	125	109	62	78	62	6387	3419	3315	18546	**3386** (ms)
46	577	1342	234	281	125	109	78	78	110	125	109	63	78	47	6397	3557	3187	18121	**3387** (ms)
47	577	1326	218	265	109	109	78	78	109	124	109	62	62	63	6618	3447	3242	18558	**3321** (ms)
48	593	1342	218	281	110	109	78	63	109	125	109	62	78	78	6442	3418	3249	18515	**3387** (ms)
49	577	1342	219	265	125	109	78	78	109	125	110	63	78	62	6438	3437	3191	18416	**3371** (ms)
50	578	1326	219	265	125	110	78	78	109	140	109	62	78	63	6428	3484	3204	18400	**3372** (ms)
AVG	**577**	**1342**	**219**	**269**	**119**	**106**	**77**	**74**	**108**	**127**	**109**	**61**	**77**	**63**	**6508**	**3459**	**3232**	**18411**	**3360** (ms)
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(µs)	(µs)	(µs)	(µs)	

Table B

SET3 Blocksize 8192B Heuristics 1024 ops

Operation no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Total
Measurement no.																
1	124	343	125	109	109	202	281	266	280	219	297	717	718	718	718	**5226** (ms)
2	94	265	78	78	93	187	249	234	249	172	234	671	670	671	686	**4631** (ms)
3	78	250	78	94	78	187	249	234	234	172	234	655	656	671	671	**4541** (ms)
4	78	266	94	78	78	187	249	234	249	171	234	671	670	671	671	**4601** (ms)
5	78	281	78	78	78	187	249	234	250	172	234	671	671	686	671	**4618** (ms)
6	78	265	78	78	78	187	249	234	250	188	218	671	671	670	687	**4602** (ms)
7	78	265	78	78	78	187	250	249	250	172	234	671	655	671	671	**4587** (ms)
8	93	265	94	78	78	187	249	234	250	172	234	671	671	671	671	**4618** (ms)
9	78	265	78	78	78	203	249	249	249	187	281	671	655	670	671	**4662** (ms)
10	78	265	94	78	94	202	250	250	250	172	265	686	686	702	686	**4758** (ms)
11	94	265	78	78	78	187	249	234	234	171	234	671	671	671	671	**4586** (ms)
12	78	281	78	78	77	187	249	250	250	172	234	671	671	671	671	**4618** (ms)
13	78	280	78	78	80	188	250	266	234	172	266	671	670	671	655	**4637** (ms)
14	94	250	94	78	78	188	250	250	250	172	234	655	671	686	671	**4621** (ms)
15	78	265	78	78	78	187	249	234	250	188	234	655	671	670	686	**4601** (ms)
16	78	250	78	78	78	188	250	250	249	171	234	670	671	671	670	**4586** (ms)
17	93	265	78	78	78	188	250	234	249	172	234	670	671	671	655	**4586** (ms)
18	78	265	93	78	78	187	249	234	234	172	234	655	671	686	671	**4585** (ms)
19	93	265	78	78	93	203	251	249	234	172	265	655	656	671	671	**4634** (ms)
20	94	265	78	78	78	188	250	234	250	171	234	671	671	671	671	**4604** (ms)
21	94	265	78	78	78	187	234	234	250	187	234	671	671	671	671	**4603** (ms)
22	78	281	78	78	93	203	249	234	234	171	234	671	671	671	670	**4616** (ms)
23	78	265	80	78	93	187	249	234	250	172	234	671	671	671	671	**4604** (ms)
24	78	265	93	78	78	187	249	265	249	171	265	670	655	671	670	**4644** (ms)
25	93	234	78	78	93	203	249	249	249	171	281	656	671	671	671	**4647** (ms)
26	94	281	78	78	94	203	266	250	249	171	234	670	655	671	670	**4664** (ms)
27	78	249	78	78	78	187	250	234	250	187	234	655	655	671	670	**4554** (ms)
28	78	250	78	78	78	188	250	234	249	187	234	670	671	671	671	**4587** (ms)
29	78	265	93	78	78	187	250	234	250	187	234	655	670	686	655	**4600** (ms)
30	93	265	94	78	78	188	250	250	250	188	281	671	686	702	686	**4760** (ms)
31	94	219	78	78	78	234	250	250	250	188	281	717	718	718	702	**4855** (ms)
32	93	249	78	78	78	203	250	250	249	171	234	671	671	670	671	**4616** (ms)
33	78	249	93	78	78	187	249	249	234	172	281	670	655	687	670	**4630** (ms)
34	94	250	78	78	78	187	249	234	250	188	234	671	671	670	671	**4603** (ms)
35	78	266	94	78	78	187	250	234	234	172	234	656	671	686	671	**4589** (ms)
36	78	250	78	78	94	188	250	234	250	172	234	671	671	671	671	**4590** (ms)
37	78	265	94	78	78	187	249	234	250	187	234	671	670	671	671	**4617** (ms)
38	93	265	78	78	78	187	250	249	250	172	234	671	655	686	671	**4617** (ms)
39	78	265	78	78	78	187	249	265	249	171	265	656	671	687	671	**4648** (ms)
40	78	265	78	78	78	187	234	234	250	188	234	671	671	686	686	**4618** (ms)
41	78	266	94	78	78	187	249	250	250	172	234	671	671	671	671	**4620** (ms)
42	94	265	93	78	78	187	249	234	234	172	234	655	671	686	671	**4601** (ms)
43	94	266	94	78	93	203	249	249	249	171	281	671	655	670	671	**4694** (ms)
44	94	250	94	78	78	188	250	234	234	171	234	670	655	671	670	**4571** (ms)
45	78	281	78	78	94	187	250	234	250	187	234	671	670	686	686	**4664** (ms)
46	78	265	94	78	437	187	234	546	592	171	234	655	655	671	670	**5567** (ms)
47	78	281	78	78	78	187	250	234	250	172	234	671	670	671	670	**4602** (ms)
48	78	265	93	78	94	203	250	234	249	187	234	655	671	670	687	**4648** (ms)
49	78	250	78	78	93	203	249	249	249	187	280	671	671	671	671	**4678** (ms)
50	78	265	78	78	78	187	234	234	249	171	234	671	655	687	687	**4586** (ms)
AVG	**85** (ms)	**264** (ms)	**84** (ms)	**79** (ms)	**90** (ms)	**192** (ms)	**249** (ms)	**248** (ms)	**254** (ms)	**177** (ms)	**245** (ms)	**669** (ms)	**669** (ms)	**678** (ms)	**674** (ms)	**4656** (ms)

Table C

SET1 Blocksize 1024B Heuristics 1024 ops

Operation no.	1	2	3	4	5	6	7	8	9	10	11	12	13	Total
Measurement no.														
1	13260	7613	7598	2215	4571	9453	16599	10436	1623	7831	1950	842	30831	**84022** (ms)
2	13182	7629	7644	2121	4337	9907	16380	10031	1638	7644	1841	843	30671	**83228** (ms)
3	13260	7597	7598	2121	4321	9781	16427	9953	1622	7785	1950	827	30169	**83272** (ms)
4	13447	7644	7597	2044	4306	9765	16489	9953	1623	8003	1888	858	31181	**83648** (ms)
5	13213	7550	7550	2216	4478	9813	16583	10031	1638	7987	1934	842	30240	**83865** (ms)
6	13198	7582	7535	2013	4321	9453	16583	10062	1638	7816	1748	842	30313	**82821** (ms)
7	13214	7598	7582	2168	4384	9594	16505	10374	1623	8034	1716	827	31350	**83650** (ms)
8	13228	7660	7597	2215	4555	9922	16520	10109	1623	7987	1762	920	29824	**84128** (ms)
9	13276	7628	7566	2215	4477	9469	16318	9813	1638	7660	1716	843	30186	**82649** (ms)
10	13197	7613	7551	2184	4618	10000	16739	10358	1622	7691	1763	827	30251	**84193** (ms)
11	13385	7738	7691	2184	4524	10108	16225	10031	1638	8549	1840	842	29922	**84785** (ms)
12	13182	7613	7613	2200	4509	9532	16302	9968	1638	7831	1810	874	31851	**83104** (ms)
13	13213	7581	7566	2169	4477	10000	16349	10359	1623	8034	1825	843	30431	**84069** (ms)
14	13291	7659	7597	2200	4634	9999	16973	10062	1607	7941	1810	843	31385	**84647** (ms)
15	13229	7644	7582	2262	4665	9485	17331	10436	1622	8206	1779	842	30554	**85114** (ms)
16	13276	7613	7582	2137	4368	9796	16458	9937	1622	8128	1857	842	60814	**83677** (ms)
17	13338	7582	7582	2184	4321	9578	16598	10046	1623	7878	1826	826	30665	**83413** (ms)
18	13261	7644	7612	2137	4446	10016	16536	9968	1653	7785	1779	827	29877	**83694** (ms)
19	13215	7613	7597	2028	4368	9703	16520	9999	1669	8221	1810	811	29781	**83584** (ms)
20	13307	7613	7550	2247	4524	9657	17488	10375	1622	8253	1794	827	30002	**85287** (ms)
21	13229	7644	7581	2216	4368	9875	16318	10093	1638	7863	1903	842	29567	**83600** (ms)
22	13276	7629	7582	2247	4352	9563	17394	10031	1606	7863	1763	843	30435	**84179** (ms)
23	13322	7645	7613	2231	4696	9391	16677	10125	1669	7909	1779	858	30711	**83946** (ms)
24	13244	7613	7551	2262	4400	9906	17050	10031	1638	8112	1856	843	33369	**84539** (ms)
25	13198	7613	7551	2199	4337	9781	16178	9922	1638	7847	1810	858	30049	**82962** (ms)
26	13276	7597	7582	2231	4493	9625	16380	10312	1638	7691	1763	873	30585	**83492** (ms)
27	13245	7613	7582	2153	4555	10155	16864	10140	1623	7987	1825	843	30100	**84615** (ms)
28	13307	7613	7582	2184	4571	9595	17332	10249	1622	7910	1747	842	32059	**84586** (ms)
29	13229	7629	7582	2168	4555	9609	17301	10015	1638	8268	1779	842	34002	**84649** (ms)
30	13229	7628	7612	2199	4337	9345	16349	9953	1638	7941	1762	843	29984	**82866** (ms)
31	13244	7628	7550	2200	4478	9906	16645	9984	1592	7753	1794	827	30437	**83631** (ms)
32	13260	7644	7582	2231	4461	9375	16271	9953	1622	8175	1716	842	30129	**83162** (ms)
33	13229	7566	7551	2215	4446	10078	16770	10046	1638	7878	1903	858	30327	**84208** (ms)
34	13245	7613	7551	2246	4477	9875	16739	10093	1606	7925	1919	826	29826	**84145** (ms)
35	13307	7628	7566	2247	4852	9485	16411	10015	1638	7940	1731	843	29416	**83692** (ms)
36	13260	7629	7597	2122	4525	9313	16161	9922	1623	8300	1857	826	29943	**83165** (ms)
37	13198	7629	7644	2106	4415	9454	16240	10016	1654	7909	1919	811	29757	**83025** (ms)
38	13244	7597	7582	2090	4321	9375	16208	9906	1591	7972	1887	842	29718	**82645** (ms)
39	13262	7598	7535	2278	4555	9703	17363	10467	1638	8362	1934	826	29892	**85551** (ms)
40	13197	7629	7582	2122	4493	9640	16380	9829	1685	7769	1872	827	33052	**83058** (ms)
41	13291	7644	7613	2215	4492	9547	16785	10046	1638	8221	1825	843	30765	**84191** (ms)
42	13244	7659	7550	2215	4540	9485	16568	10359	1638	7924	1934	842	30265	**83988** (ms)
43	13228	7598	7551	2246	4446	9563	16770	10187	1622	7816	1918	858	30684	**83834** (ms)
44	13261	7582	7582	2028	4399	9875	16536	10219	1638	7878	1857	842	30395	**83727** (ms)
45	13198	7613	7629	2075	4321	9407	16786	10249	1622	8331	1887	874	29686	**84022** (ms)
46	13229	7598	7550	2137	4274	9453	16334	10031	1622	7925	1872	827	30859	**82883** (ms)
47	13214	7582	7535	2246	4368	9641	16848	10155	1638	8128	1919	826	30665	**84131** (ms)
48	13198	7613	7519	2277	4493	9672	16458	10155	1622	8206	1872	842	29402	**83956** (ms)
49	13245	7597	7581	2169	4290	9328	16318	10030	1638	7940	1794	811	30223	**82771** (ms)
50	13354	7644	7628	2230	4617	10093	17284	10359	1638	8518	1779	843	30660	**86018** (ms)
AVG	**13253**	**7619**	**7582**	**2181**	**4463**	**9683**	**16633**	**10104**	**1631**	**7991**	**1831**	**841**	**31145**	**83842** (ms)
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(µs)	

Table D

SET2 Blocksize 1024B Heuristics 1024 ops

Operation no. / Measurement no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	Total
1	2699	3229	920	1108	250	281	250	265	203	234	203	109	125	109	11095	337185	4826	118546	**10457** (ms)
2	1918	2418	952	1092	218	265	250	281	188	218	187	109	110	110	10930	339556	5200	117334	**8789** (ms)
3	1935	2449	920	1123	218	266	234	266	203	218	188	94	110	94	10877	338397	4773	118680	**8791** (ms)
4	1981	2371	951	1248	218	265	266	265	187	218	188	94	109	109	10634	339093	4702	115741	**8940** (ms)
5	2012	2308	921	1108	219	265	265	265	188	218	187	93	109	94	10812	336159	4778	119454	**8723** (ms)
6	1936	2372	936	1092	218	265	265	265	203	218	187	109	109	94	10949	337190	4732	119493	**8741** (ms)
7	1965	2308	951	1107	219	265	234	265	203	218	188	93	109	94	12125	335839	4738	117904	**8690** (ms)
8	1919	2325	921	1108	219	265	234	265	187	218	187	93	109	94	10781	336415	4799	121452	**8617** (ms)
9	2012	2402	951	1092	218	265	249	265	188	250	188	94	109	109	16702	336909	4759	119405	**8870** (ms)
10	1935	2356	905	1092	218	265	280	265	187	219	172	94	110	94	11030	337348	4793	124093	**8669** (ms)
11	1950	2402	982	1108	203	266	265	265	187	234	172	94	109	93	11100	336281	4796	118238	**8800** (ms)
12	1934	2371	921	1092	218	266	250	281	187	218	187	94	110	94	11013	337638	4818	121199	**8698** (ms)
13	1919	2434	921	1092	219	265	250	280	187	218	187	109	124	109	10721	338861	4819	118074	**8786** (ms)
14	1919	2434	920	1092	218	265	249	265	187	219	172	94	109	93	10975	336646	4718	122124	**8710** (ms)
15	1950	2325	967	1107	219	265	281	265	187	218	187	94	109	93	10832	339599	4808	118429	**8741** (ms)
16	1982	2418	921	1107	219	265	280	265	187	219	187	109	125	109	10894	337200	4794	121171	**8867** (ms)
17	1950	2356	952	1093	218	265	249	265	187	219	187	93	109	109	10857	337450	4856	117994	**8723** (ms)
18	1997	2418	905	1108	218	265	249	265	187	234	187	93	109	94	10988	336278	4764	116405	**8797** (ms)
19	1935	2387	952	1092	218	265	250	266	187	234	280	109	125	109	10765	341085	5101	112935	**8879** (ms)
20	1935	2371	920	1108	219	265	281	265	188	218	172	109	125	109	10732	337818	4774	123195	**8762** (ms)
21	1903	2309	968	1108	218	265	249	265	187	219	203	94	110	94	11102	338566	4725	115869	**8662** (ms)
22	1903	2309	905	1107	218	265	250	265	188	218	187	109	125	110	11109	336284	4741	118513	**8630** (ms)
23	1919	2325	967	1108	203	266	234	266	187	234	187	110	125	109	10865	337518	4841	118393	**8712** (ms)
24	1919	2386	905	1092	218	281	249	265	188	344	203	109	124	109	10645	337275	4847	118286	**8863** (ms)
25	1919	2325	920	1092	203	266	234	266	187	218	188	110	125	109	10879	339125	4875	118984	**8636** (ms)
26	1935	2325	920	1107	203	281	250	265	187	218	171	93	109	94	11004	337187	4811	118934	**8630** (ms)
27	1918	2371	952	1107	218	265	249	281	187	219	187	93	109	94	10952	336938	5144	117531	**8721** (ms)
28	1934	2293	983	1108	218	265	249	265	188	266	203	109	124	109	10948	337566	4810	118919	**8786** (ms)
29	1934	2356	921	1107	218	265	250	281	188	218	187	94	110	94	10923	337445	4884	120912	**8697** (ms)
30	1919	2324	936	1139	218	265	250	265	187	266	187	109	125	109	10691	337134	4790	116483	**8768** (ms)
31	1966	2402	1186	1092	218	265	250	266	188	218	203	110	125	109	10953	338394	4805	118789	**9071** (ms)
32	1904	2324	936	1107	203	265	265	265	187	219	187	109	125	109	10895	338204	4806	119163	**8678** (ms)
33	1903	2371	920	1092	218	265	296	281	187	219	187	109	109	94	11013	337518	4934	123031	**8727** (ms)
34	1887	2309	952	1092	219	265	249	265	187	219	187	93	109	110	10706	337669	4755	118337	**8614** (ms)
35	1919	2403	921	1108	218	265	250	265	187	219	187	109	125	109	10948	336927	4787	122812	**8760** (ms)
36	1934	2325	952	1107	218	265	249	265	187	219	265	93	109	94	10958	338256	4826	117944	**8754** (ms)
37	1906	2387	921	1092	218	265	250	265	188	218	187	109	110	94	10964	337548	4767	120381	**8684** (ms)
38	1903	2356	967	1108	219	265	250	265	171	219	187	109	110	94	10739	339884	4773	119113	**8698** (ms)
39	1903	2465	921	1092	218	265	265	265	171	218	187	94	109	93	10781	336513	4766	120699	**8739** (ms)
40	1888	2309	951	1107	202	265	250	265	187	219	265	109	124	109	10934	339314	4832	118120	**8723** (ms)
41	1919	2325	905	1108	203	266	249	265	327	218	187	109	124	109	10940	337610	4810	120228	**8788** (ms)
42	1888	2309	951	1107	218	265	250	266	203	218	187	94	124	109	10687	338250	4794	119354	**8662** (ms)
43	1872	2309	920	1092	218	265	265	265	188	218	187	94	110	94	11021	337242	4813	119975	**8570** (ms)
44	1888	2308	905	1123	218	266	250	266	187	218	203	110	109	93	10681	337276	4990	119705	**8617** (ms)
45	1934	2309	952	1124	218	265	250	265	188	234	203	110	109	109	15396	337620	4779	118941	**8747** (ms)
46	1919	2387	921	1107	218	265	250	266	187	234	203	110	109	93	10889	337666	4791	122890	**8745** (ms)
47	1872	2340	967	1092	218	265	250	265	187	218	187	94	109	93	10781	336636	4753	118592	**8628** (ms)
48	1919	2294	920	1107	219	265	280	265	203	219	187	93	109	110	10732	337598	4761	119437	**8663** (ms)
49	1919	2277	967	1107	218	265	266	265	187	218	234	109	125	109	10988	340321	4788	118257	**8740** (ms)
50	1887	2308	920	1123	218	266	234	266	296	218	187	109	124	93	10797	341841	5082	121859	**8729** (ms)
AVG	**1942**	**2372**	**940**	**1107**	**217**	**266**	**254**	**267**	**194**	**226**	**194**	**102**	**115**	**101**	**11115**	**337805**	**4825**	**119246**	**8770** (ms)
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(µs)	(µs)	(µs)	(µs)	

Table E

SET3 Blocksize 1024B Heuristics 1024 ops

Operation no. / Measurement no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Total
1	156	765	140	171	172	140	140	141	156	140	203	188	250	219	187	**3168** (ms)
2	156	767	141	141	156	141	125	140	140	140	156	156	156	172	172	**2859** (ms)
3	140	764	141	141	140	125	125	125	125	124	156	156	171	172	172	**2777** (ms)
4	156	764	141	140	140	140	125	125	141	141	171	172	172	156	156	**2840** (ms)
5	141	764	140	141	141	125	125	140	124	125	156	156	172	171	172	**2793** (ms)
6	140	765	141	156	141	140	141	141	140	140	172	172	171	156	156	**2872** (ms)
7	141	767	141	141	140	140	125	125	125	140	156	156	171	172	172	**2812** (ms)
8	141	764	125	140	141	140	125	125	124	140	172	172	171	203	171	**2854** (ms)
9	140	795	141	140	141	125	125	125	124	125	156	156	156	172	171	**2792** (ms)
10	156	780	140	141	140	141	124	125	140	141	172	171	172	172	171	**2886** (ms)
11	141	749	140	140	141	141	140	140	140	141	156	171	172	156	156	**2824** (ms)
12	140	780	141	140	140	124	125	141	125	125	156	156	156	172	171	**2792** (ms)
13	156	797	140	140	141	125	125	124	125	140	156	171	172	156	156	**2824** (ms)
14	140	749	140	140	141	125	125	125	125	125	156	156	171	172	171	**2761** (ms)
15	156	764	141	140	140	125	124	141	141	125	156	156	156	171	172	**2808** (ms)
16	140	765	140	140	140	140	125	125	125	141	171	172	172	156	156	**2808** (ms)
17	140	764	156	141	141	141	125	124	125	140	172	172	156	156	156	**2809** (ms)
18	140	749	125	141	141	125	125	125	124	125	156	156	171	171	172	**2746** (ms)
19	140	764	140	141	141	141	125	140	124	125	156	156	156	156	171	**2776** (ms)
20	140	796	141	141	140	140	124	141	141	140	156	156	156	171	172	**2855** (ms)
21	140	780	140	140	141	125	141	125	125	140	171	172	171	172	156	**2839** (ms)
22	140	765	156	140	140	140	125	125	125	141	171	172	172	172	156	**2840** (ms)
23	140	748	156	140	141	141	141	124	125	140	172	172	171	172	156	**2839** (ms)
24	140	764	140	140	140	156	125	124	140	140	172	171	172	156	172	**2852** (ms)
25	140	765	141	140	140	140	125	125	140	124	156	156	156	156	156	**2760** (ms)
26	141	749	156	140	140	140	124	125	125	125	156	171	172	171	156	**2791** (ms)
27	156	764	141	141	141	141	125	124	125	140	172	171	172	156	156	**2825** (ms)
28	156	764	141	140	141	140	125	140	125	124	156	156	156	172	172	**2808** (ms)
29	141	765	140	140	140	156	140	140	140	141	156	156	156	172	171	**2854** (ms)
30	156	749	141	141	156	141	125	140	140	124	156	156	156	171	172	**2824** (ms)
31	140	733	141	141	141	141	125	125	125	124	156	156	156	156	172	**2732** (ms)
32	140	764	140	141	141	141	140	140	124	125	156	156	171	172	156	**2807** (ms)
33	156	765	140	140	140	125	124	125	125	125	156	156	172	171	172	**2792** (ms)
34	140	811	140	141	141	141	125	125	140	125	156	156	171	172	156	**2840** (ms)
35	140	733	140	141	141	141	125	125	124	125	156	156	171	172	171	**2761** (ms)
36	140	780	140	156	140	124	125	125	125	141	171	172	156	156	156	**2807** (ms)
37	141	764	140	140	156	140	124	141	141	140	156	156	156	156	172	**2823** (ms)
38	141	765	140	140	140	156	125	140	140	141	156	171	203	156	156	**2870** (ms)
39	141	780	141	140	140	140	125	141	140	140	156	156	156	156	156	**2808** (ms)
40	141	780	141	140	141	125	125	141	141	140	156	172	171	172	172	**2858** (ms)
41	141	749	141	140	140	156	125	140	141	141	171	156	156	171	172	**2840** (ms)
42	140	764	156	140	140	140	125	125	125	124	156	156	156	156	172	**2775** (ms)
43	141	765	141	140	141	140	140	125	141	141	171	172	171	172	172	**2873** (ms)
44	141	749	141	141	140	141	141	124	140	140	172	171	156	156	156	**2809** (ms)
45	141	795	140	140	141	125	125	124	141	141	156	156	172	156	156	**2809** (ms)
46	140	749	141	140	140	140	125	125	125	125	156	156	172	172	172	**2778** (ms)
47	141	796	140	140	140	156	140	125	140	141	172	171	172	171	156	**2901** (ms)
48	140	765	141	140	141	140	125	125	124	125	156	156	187	171	172	**2808** (ms)
49	140	780	140	140	140	124	125	125	125	125	156	156	188	172	171	**2807** (ms)
50	141	796	141	141	141	125	125	140	140	141	156	156	156	156	156	**2811** (ms)
AVG	**144** (ms)	**767** (ms)	**141** (ms)	**142** (ms)	**142** (ms)	**137** (ms)	**128** (ms)	**131** (ms)	**132** (ms)	**134** (ms)	**162** (ms)	**163** (ms)	**168** (ms)	**167** (ms)	**165** (ms)	**2824** (ms)

Table F

SET1 Blocksize 65536B Heuristics 1024 ops

Operation no. / Measurement no.	1	2	3	4	5	6	7	8	9	10	11	12	13	Total
1	6084	1638	1623	3604	1201	1497	2074	827	765	6958	905	484	9237	**27669** (ms)
2	6755	1622	1638	3572	1216	1435	2012	811	765	6973	936	468	9230	**28212** (ms)
3	5975	1623	1622	3386	1186	1435	2028	795	733	7597	936	452	9307	**27777** (ms)
4	5959	1623	1623	3557	1202	1435	2012	811	748	7191	936	468	9200	**27574** (ms)
5	5959	1638	1638	3541	1202	1435	2028	827	749	7083	951	468	9253	**27528** (ms)
6	6084	1638	1623	3604	1201	1451	2028	811	764	7223	858	468	9174	**27762** (ms)
7	6068	1638	1669	3525	1217	1435	2028	812	749	7176	951	468	9352	**27745** (ms)
8	6006	1669	1622	3557	1201	1450	2012	826	765	7161	920	468	9346	**27666** (ms)
9	6022	1622	1622	3479	1201	1450	2043	795	764	7410	905	468	9049	**27790** (ms)
10	5990	1638	1622	3400	1201	1435	2028	811	780	7098	920	452	9252	**27384** (ms)
11	6038	1607	1623	3572	1201	1451	2059	827	765	7239	905	468	9303	**27764** (ms)
12	5974	1622	1607	3463	1201	1435	2012	812	764	7285	827	468	9208	**27479** (ms)
13	6022	1638	1638	3573	1201	1435	2013	827	748	7191	858	437	9187	**27590** (ms)
14	5975	1607	1622	3432	1186	1435	2043	811	764	6942	936	452	9459	**27214** (ms)
15	6022	1622	1622	3573	1216	1451	2012	812	749	6927	920	468	9316	**27403** (ms)
16	6022	1638	1623	3650	1217	1451	2044	827	780	7379	921	468	9488	**28029** (ms)
17	6006	1622	1638	3620	1248	1436	2028	811	764	7239	889	468	9215	**27778** (ms)
18	5975	1623	1607	3494	1217	1435	2028	827	749	7113	952	468	9173	**27497** (ms)
19	6022	1623	1623	3557	1201	1435	2013	811	764	7379	936	468	9155	**27841** (ms)
20	5975	1638	1622	3541	1202	1451	2044	827	749	7363	920	468	9145	**27809** (ms)
21	6037	1622	1638	3588	1201	1451	2028	811	749	7223	858	468	9120	**27683** (ms)
22	5975	1638	1607	3588	1217	1451	2044	811	764	7052	983	468	9520	**27608** (ms)
23	6006	1623	1622	3557	1201	1450	2028	811	765	7145	921	468	9183	**27606** (ms)
24	5944	1638	1638	3588	1217	1436	2059	811	733	7114	920	483	9364	**27590** (ms)
25	5974	1622	1638	3588	1186	1451	2028	811	764	7238	920	484	9158	**27713** (ms)
26	5991	1622	1607	3556	1185	1435	2028	811	749	6942	1014	483	9491	**27432** (ms)
27	5990	1606	1638	3573	1217	1435	2043	811	749	7893	936	468	9117	**28368** (ms)
28	5959	1623	1623	3526	1201	1451	2044	827	764	7005	905	468	15416	**27411** (ms)
29	5975	1638	1638	3573	1201	1450	2028	827	749	7379	920	468	9547	**27856** (ms)
30	6069	1623	1623	3494	1201	1435	2044	827	749	7207	936	468	9419	**27685** (ms)
31	6006	1622	1638	3573	1201	1435	2028	811	765	7191	936	468	9304	**27683** (ms)
32	5959	1623	1623	3541	1217	1435	2028	811	765	7816	936	468	9292	**28231** (ms)
33	6006	1623	1638	3604	1233	1435	2028	827	749	7254	905	468	9453	**27779** (ms)
34	5944	1607	1622	3463	1186	1451	2028	827	749	7145	952	452	9507	**27436** (ms)
35	6021	1622	1622	3573	1217	1435	2044	811	765	7254	952	468	9241	**27793** (ms)
36	5991	1622	1607	3495	1217	1435	2013	811	765	7613	874	483	9556	**27936** (ms)
37	6037	1607	1638	3494	1202	1435	2012	796	765	7067	843	452	9428	**27357** (ms)
38	5991	1623	1622	3510	1186	1435	2044	811	749	7364	952	453	9121	**27749** (ms)
39	6022	1670	1622	3557	1201	1435	2028	811	764	7208	983	468	9156	**27778** (ms)
40	6006	1669	1622	3464	1201	1435	2028	811	733	7426	936	468	9474	**27808** (ms)
41	5975	1623	1622	3557	1202	1466	2012	811	796	7113	889	468	9451	**27543** (ms)
42	5975	1607	1622	3416	1201	1436	2059	811	749	7114	967	452	9350	**27418** (ms)
43	5990	1622	1622	3573	1217	1451	2028	811	764	7207	936	468	9358	**27698** (ms)
44	5990	1638	1638	3448	1217	1482	2028	811	764	7176	842	468	9520	**27512** (ms)
45	6006	1638	1638	3619	1201	1435	2044	812	764	7925	921	468	9365	**28480** (ms)
46	6006	1623	1607	3354	1217	1466	2028	796	749	7285	951	468	9224	**27559** (ms)
47	6039	1622	1623	3572	1201	1436	2044	811	764	7035	951	468	9207	**27575** (ms)
48	5975	1623	1607	3557	1202	1435	2059	811	749	7035	920	468	9394	**27450** (ms)
49	6006	1623	1638	3572	1217	1451	2028	811	764	7239	858	453	9614	**27670** (ms)
50	5991	1638	1622	3526	1232	1435	2044	812	765	6942	936	452	9218	**27404** (ms)
AVG	**6016**	**1628**	**1626**	**3536**	**1206**	**1444**	**2032**	**814**	**758**	**7231**	**921**	**466**	**9432**	**27686** (ms)
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(µs)	

Table G

SET2 Blocksize 65536B Heuristics 1024 ops

Operation no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	Total
Measurement no.																			
1	484	4072	203	281	141	125	78	78	125	124	110	63	78	62	6327	3156	3777	5937	**6043** (ms)
2	468	4072	218	249	125	109	78	78	93	125	93	46	62	46	6013	3099	3079	5567	**5880** (ms)
3	499	4041	218	249	125	125	78	78	94	109	94	47	63	47	6091	3204	3020	5411	**5885** (ms)
4	453	4072	219	250	125	109	78	78	93	109	94	47	62	62	6158	3084	3072	5551	**5869** (ms)
5	483	4087	203	249	124	109	78	78	94	125	94	47	63	47	6179	3150	3085	5553	**5899** (ms)
6	483	4040	219	265	124	109	78	78	94	125	94	47	63	47	6137	3201	3118	5401	**5884** (ms)
7	453	4072	218	249	125	109	62	78	94	109	93	46	62	46	5988	3137	3098	5465	**5834** (ms)
8	468	4072	218	249	125	110	78	78	93	109	94	47	63	62	6346	3146	3083	5723	**5884** (ms)
9	452	4072	219	250	109	109	78	78	94	109	93	46	62	46	6119	3104	3083	5475	**5835** (ms)
10	468	4072	203	250	125	110	78	78	93	109	93	46	62	62	5974	3102	3023	5491	**5867** (ms)
11	453	4056	219	265	124	93	78	78	94	109	93	62	78	62	6140	3124	3048	5793	**5882** (ms)
12	483	4056	203	250	125	109	78	78	93	125	93	47	62	47	6125	3139	3042	5427	**5867** (ms)
13	452	4072	218	265	125	109	78	78	93	109	109	62	62	46	6039	3131	3024	5568	**5896** (ms)
14	453	4056	202	249	109	110	78	78	94	109	94	47	78	62	6262	3183	3036	5742	**5837** (ms)
15	453	4056	219	250	124	109	78	78	94	109	109	62	63	62	6030	3144	3067	5747	**5884** (ms)
16	468	4071	218	265	125	94	78	78	94	109	109	62	62	47	6156	3082	3024	5432	**5898** (ms)
17	500	4056	218	265	125	109	78	78	94	110	94	47	63	47	6235	3135	3089	5451	**5902** (ms)
18	500	4072	218	265	125	109	78	78	93	109	93	63	78	63	6186	3151	3025	5517	**5962** (ms)
19	468	4072	219	250	125	109	78	78	93	125	93	62	62	62	6088	3128	3024	5479	**5914** (ms)
20	468	4040	218	249	125	94	78	78	94	110	109	62	63	62	6032	3131	3013	5764	**5868** (ms)
21	452	4087	218	249	109	110	78	78	94	109	94	47	63	47	6075	3192	3074	5530	**5853** (ms)
22	468	4040	219	265	124	109	78	78	94	109	109	62	78	62	6069	3111	3078	5696	**5913** (ms)
23	483	4087	202	249	109	110	78	78	94	124	94	47	63	47	6201	3136	3082	5445	**5883** (ms)
24	468	4071	219	250	109	109	78	78	94	125	110	47	62	47	6116	3091	3086	5490	**5885** (ms)
25	452	4071	218	249	125	110	62	78	94	109	94	47	77	63	6148	3118	3084	5417	**5867** (ms)
26	452	4072	219	249	124	109	78	93	94	109	93	46	62	62	6064	3116	3006	5417	**5880** (ms)
27	483	4119	203	250	125	109	78	78	93	109	93	46	62	46	6108	3141	3076	5409	**5912** (ms)
28	452	4087	218	250	110	109	62	78	93	109	93	47	62	47	6011	3179	3010	5671	**5835** (ms)
29	452	4040	218	250	125	109	78	63	94	109	93	47	62	47	6248	3158	3041	5491	**5805** (ms)
30	484	4072	218	250	110	109	62	78	94	110	94	47	63	47	6257	3153	3006	5470	**5856** (ms)
31	453	4056	202	249	125	110	78	78	94	109	93	47	78	63	6083	3106	3233	5730	**5853** (ms)
32	468	4071	219	250	125	93	78	78	94	109	94	47	62	47	6117	3102	3029	5479	**5853** (ms)
33	452	4041	219	266	125	109	78	78	93	109	94	47	62	47	6120	3152	3131	5718	**5838** (ms)
34	484	4056	219	250	109	109	63	78	93	109	94	62	78	62	6101	3207	3006	5401	**5884** (ms)
35	483	4056	218	250	125	109	78	78	93	125	94	47	62	47	6089	3070	3019	5418	**5883** (ms)
36	453	4056	218	250	125	109	78	78	93	109	94	47	62	47	6178	3124	3087	5380	**5837** (ms)
37	453	4072	218	250	125	109	78	78	93	109	94	47	63	47	6152	3140	3058	5429	**5854** (ms)
38	468	4040	202	249	125	110	62	78	94	109	94	63	78	63	6239	3092	3091	5439	**5853** (ms)
39	468	4088	203	250	125	109	78	78	93	109	94	47	63	47	6162	3103	3041	5560	**5870** (ms)
40	453	4040	218	249	125	110	78	78	94	109	94	47	62	63	6336	3094	3018	5478	**5838** (ms)
41	453	4072	218	250	125	109	78	78	93	109	93	46	62	46	6087	3092	3031	5496	**5850** (ms)
42	453	4087	218	265	125	109	78	78	93	109	93	47	62	47	6024	3079	3051	5489	**5882** (ms)
43	468	4056	219	266	125	109	78	78	94	109	94	47	62	47	6161	3112	3018	5485	**5870** (ms)
44	453	4040	218	250	125	94	78	62	94	110	94	47	63	47	6414	3092	3037	5451	**5793** (ms)
45	499	4056	219	250	125	109	78	78	94	110	94	47	63	47	6151	3171	3016	5381	**5887** (ms)
46	468	4056	203	249	125	110	78	78	109	109	93	47	62	62	6082	3066	3058	5424	**5867** (ms)
47	500	4056	219	265	124	109	78	78	93	109	93	46	62	46	6166	3107	3079	5654	**5896** (ms)
48	452	4041	218	249	109	110	78	78	94	109	109	63	78	63	6122	3094	3047	5724	**5869** (ms)
49	468	4071	203	250	109	109	78	78	94	110	94	47	63	47	6088	3147	3002	5485	**5839** (ms)
50	452	4088	219	250	109	109	63	78	93	109	94	47	63	47	6059	3096	3024	5765	**5839** (ms)
AVG	**467**	**4064**	**215**	**254**	**122**	**108**	**76**	**78**	**95**	**112**	**96**	**51**	**66**	**53**	**6137**	**3127**	**3069**	**5538**	**5873** (ms)
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(µs)	(µs)	(µs)	(µs)	

Table H

SET3 Blocksize 65536B Heuristics 1024 ops

Operation no. / Measurement no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Total
1	109	250	109	93	124	219	281	265	265	203	297	202	265	281	265	**3228** (ms)
2	109	249	93	109	141	219	297	281	280	218	296	219	296	281	328	**3416** (ms)
3	109	249	93	93	141	219	265	265	266	203	265	203	296	281	312	**3260** (ms)
4	109	249	94	94	125	203	266	250	265	187	281	203	281	265	312	**3184** (ms)
5	94	234	109	93	125	203	265	265	249	203	266	203	281	281	312	**3183** (ms)
6	110	250	94	94	141	218	281	265	280	203	281	202	296	281	312	**3308** (ms)
7	109	234	94	94	125	218	265	265	265	203	265	203	266	281	312	**3199** (ms)
8	93	265	94	94	125	202	281	281	281	203	312	203	281	265	281	**3261** (ms)
9	94	265	93	109	125	219	280	265	265	203	281	202	281	281	312	**3275** (ms)
10	109	250	94	94	140	234	296	281	280	203	281	219	281	281	312	**3355** (ms)
11	93	249	93	93	140	234	296	297	297	218	281	218	312	297	328	**3446** (ms)
12	94	250	109	109	124	218	281	281	266	203	312	218	281	266	281	**3293** (ms)
13	109	250	94	94	125	202	265	250	266	203	265	203	265	265	312	**3168** (ms)
14	94	249	93	93	124	203	265	280	265	203	296	203	265	265	281	**3179** (ms)
15	109	250	94	94	125	219	280	265	266	203	281	203	296	266	312	**3263** (ms)
16	109	234	94	110	125	218	281	265	281	203	281	203	281	280	312	**3277** (ms)
17	93	250	94	94	140	218	281	266	265	202	265	203	280	281	312	**3244** (ms)
18	109	249	93	93	124	203	281	265	265	203	281	218	281	265	280	**3210** (ms)
19	109	250	94	94	125	218	265	265	265	203	266	203	281	266	312	**3216** (ms)
20	109	250	94	94	125	202	265	265	265	188	265	203	265	265	312	**3167** (ms)
21	109	250	109	93	125	203	265	265	265	203	281	203	281	265	312	**3229** (ms)
22	109	234	93	94	125	219	281	265	265	203	265	203	266	265	312	**3199** (ms)
23	109	234	93	93	124	203	265	265	265	203	281	218	296	265	312	**3226** (ms)
24	110	250	94	94	125	202	265	266	266	203	265	203	265	265	312	**3185** (ms)
25	109	250	94	94	140	203	266	265	265	187	297	187	265	265	265	**3152** (ms)
26	110	250	110	94	125	202	265	266	266	203	265	203	281	265	312	**3217** (ms)
27	109	249	109	93	124	205	265	265	265	203	265	203	266	265	280	**3166** (ms)
28	109	249	93	94	141	202	265	281	249	202	297	203	250	265	265	**3165** (ms)
29	94	249	93	93	124	218	280	281	281	203	297	218	312	297	328	**3368** (ms)
30	109	234	109	109	140	234	296	280	281	218	328	218	281	296	296	**3429** (ms)
31	93	249	94	94	125	219	265	280	281	219	312	219	265	281	281	**3277** (ms)
32	109	250	94	94	125	234	296	296	296	218	296	219	297	296	343	**3463** (ms)
33	109	234	94	94	125	203	265	265	265	203	265	203	297	280	312	**3214** (ms)
34	109	250	94	94	140	219	281	280	281	219	312	203	265	281	281	**3309** (ms)
35	109	265	94	93	125	203	265	281	266	203	297	203	265	265	281	**3215** (ms)
36	109	249	93	93	125	218	281	281	281	203	281	202	296	280	328	**3320** (ms)
37	109	249	109	93	124	218	280	281	281	219	328	218	280	296	296	**3381** (ms)
38	110	234	94	94	140	218	265	266	265	202	265	203	296	281	312	**3245** (ms)
39	109	249	93	93	125	203	281	281	265	203	266	203	281	281	296	**3229** (ms)
40	109	250	109	93	124	218	281	265	281	219	280	218	296	281	328	**3352** (ms)
41	94	249	109	94	125	219	281	281	281	219	281	218	296	281	296	**3324** (ms)
42	109	250	94	94	140	218	281	266	281	218	281	203	297	281	312	**3325** (ms)
43	109	250	94	94	140	218	281	265	265	203	281	218	281	266	312	**3277** (ms)
44	109	250	94	94	125	218	281	265	281	203	281	203	297	280	312	**3293** (ms)
45	109	249	93	93	125	203	281	266	281	202	312	218	281	281	280	**3274** (ms)
46	94	234	93	109	125	219	281	265	265	203	265	203	266	281	312	**3215** (ms)
47	110	250	110	94	125	218	281	265	281	202	281	218	296	265	312	**3308** (ms)
48	109	265	94	109	140	218	281	281	265	203	296	218	265	265	281	**3290** (ms)
49	109	234	109	94	125	219	280	265	266	203	280	203	280	281	328	**3276** (ms)
50	110	234	94	94	125	218	296	296	296	219	312	219	296	296	280	**3385** (ms)
AVG	**106** (ms)	**247** (ms)	**97** (ms)	**96** (ms)	**129** (ms)	**214** (ms)	**277** (ms)	**271** (ms)	**272** (ms)	**205** (ms)	**285** (ms)	**208** (ms)	**282** (ms)	**276** (ms)	**304** (ms)	**3269** (ms)

Table I

SET1 Blocksize 65536B Heuristics 128 ops

Operation no. / Measurement no.	1	2	3	4	5	6	7	8	9	10	11	12	13	Total
1	5991	1622	1622	3448	1342	1716	2152	796	749	7176	967	453	9420	**28043** (ms)
2	5993	1638	1622	3354	1217	1451	2044	812	749	6911	937	484	9263	**27221** (ms)
3	5974	1606	1638	3604	1232	1450	2028	827	765	7176	842	468	9102	**27619** (ms)
4	5960	1622	1638	3401	1186	1436	2044	811	764	6942	936	452	9407	**27201** (ms)
5	6021	1638	1607	3588	1201	1435	2013	827	764	7316	905	468	9313	**27792** (ms)
6	5975	1623	1623	3385	1185	1450	2028	811	749	7255	936	484	11844	**27516** (ms)
7	6022	1638	1622	3604	1217	1451	2013	811	764	7036	936	468	9206	**27591** (ms)
8	6053	1638	1638	3557	1201	1451	2043	812	749	7207	905	469	9228	**27732** (ms)
9	6053	1607	1638	3572	1216	1435	2028	811	734	7597	889	468	9507	**28058** (ms)
10	5990	1622	1623	3385	1201	1435	2028	811	749	7067	951	452	9325	**27323** (ms)
11	6115	1622	1623	3557	1202	1466	2012	811	749	7144	827	484	9349	**27621** (ms)
12	5990	1607	1638	3494	1186	1435	2028	811	764	7394	952	453	9509	**27762** (ms)
13	6133	1638	1623	3510	1201	1450	2028	811	765	6911	936	483	9229	**27498** (ms)
14	6006	1638	1607	3464	1217	1435	2044	811	765	7161	967	468	9299	**27592** (ms)
15	6038	1638	1622	3541	1201	1451	2012	827	764	7332	842	468	9140	**27745** (ms)
16	5991	1622	1622	3400	1186	1451	2044	811	749	6942	936	452	9217	**27215** (ms)
17	6100	1638	1638	3604	1202	1451	2028	827	749	7098	920	468	9294	**27732** (ms)
18	6006	1623	1622	3603	1201	1467	2044	827	764	6833	982	484	9401	**27465** (ms)
19	6100	1623	1623	3604	1217	1451	2028	811	749	7176	951	468	9216	**27810** (ms)
20	5975	1623	1622	3525	1201	1451	2044	811	764	7114	952	468	9281	**27559** (ms)
21	5975	1622	1622	3572	1217	1435	2012	827	733	7535	858	453	9116	**27870** (ms)
22	6006	1623	1622	3401	1186	1435	2028	811	749	7441	936	452	9369	**27699** (ms)
23	6022	1622	1622	3541	1201	1435	2012	827	765	7207	936	468	9122	**27667** (ms)
24	6038	1638	1622	3526	1217	1435	2043	827	749	7488	936	483	9317	**28011** (ms)
25	5975	1622	1638	3619	1201	1451	2028	827	733	7192	936	483	9255	**27714** (ms)
26	6084	1623	1638	3494	1201	1451	2028	811	749	7254	951	468	9265	**27761** (ms)
27	5991	1622	1623	3495	1202	1451	2013	795	764	6974	842	468	9139	**27249** (ms)
28	5943	1623	1607	3432	1186	1450	2043	811	749	7098	936	453	9185	**27340** (ms)
29	6100	1607	1622	3604	1217	1420	2028	811	749	6896	936	453	9341	**27452** (ms)
30	5991	1622	1623	3588	1217	1513	2075	811	764	7457	952	468	9317	**28090** (ms)
31	5959	1638	1622	3572	1201	1435	2028	811	765	7192	905	483	9403	**27620** (ms)
32	6021	1622	1622	3447	1185	1435	2028	811	764	7160	905	468	9508	**27478** (ms)
33	6100	1638	1638	3604	1201	1451	2013	811	749	7066	905	468	9141	**27653** (ms)
34	6021	1606	1622	3557	1217	1435	2012	811	765	7301	952	453	9299	**27761** (ms)
35	5944	1638	1622	3448	1217	1435	2028	827	749	7191	920	468	9463	**27496** (ms)
36	6131	1638	1638	3572	1202	1420	2043	811	749	7831	843	483	9795	**28371** (ms)
37	5991	1622	1623	3416	1185	1482	2043	796	765	6989	889	468	9581	**27279** (ms)
38	6006	1623	1638	3557	1202	1435	2028	811	764	7223	936	468	10279	**27701** (ms)
39	5990	1638	1638	3510	1186	1450	2028	795	749	7160	951	468	9667	**27573** (ms)
40	6053	1623	1638	3557	1232	1451	2013	811	764	6817	858	468	9226	**27294** (ms)
41	5961	1638	1622	3448	1170	1435	2028	811	765	7582	905	468	9186	**27842** (ms)
42	6037	1623	1623	3541	1201	1435	2028	811	765	7114	951	452	9258	**27590** (ms)
43	5960	1623	1638	3510	1201	1436	2059	811	749	7239	952	483	9149	**27670** (ms)
44	5991	1623	1622	3604	1201	1450	2028	811	749	6958	905	468	9546	**27420** (ms)
45	5974	1622	1623	3479	1217	1436	2044	811	733	7005	905	468	9099	**27326** (ms)
46	6021	1622	1638	3603	1233	1435	2012	811	749	7083	967	468	9221	**27651** (ms)
47	5975	1638	1638	3588	1202	1435	2059	812	749	7254	951	484	9330	**27794** (ms)
48	6006	1622	1623	3588	1201	1451	2028	827	764	6958	842	484	15459	**27409** (ms)
49	6021	1622	1623	3495	1217	1435	2043	827	765	7067	874	468	9558	**27467** (ms)
50	6068	1623	1638	3572	1185	1450	2028	811	749	7847	920	468	9254	**28368** (ms)
AVG	**6017** (ms)	**1626** (ms)	**1627** (ms)	**3523** (ms)	**1206** (ms)	**1450** (ms)	**2033** (ms)	**814** (ms)	**755** (ms)	**7187** (ms)	**918** (ms)	**468** (ms)	**9508** (µs)	**27634** (ms)

Table J

SET2 Blocksize 65536B Heuristics 128 ops

Operation no. / Measurement no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	Total
1	468	1045	218	265	140	109	78	78	94	109	94	62	62	62	6056	3140	3029	5626	**2902** (ms)
2	468	1060	203	250	125	93	78	78	93	109	93	47	62	47	6198	3240	3064	5512	**2824** (ms)
3	484	1045	218	249	125	94	78	78	94	109	93	62	63	47	6080	3192	3058	5484	**2857** (ms)
4	468	1060	219	266	125	78	78	78	94	109	93	63	63	47	6007	3249	3106	5514	**2859** (ms)
5	468	1077	218	265	125	78	78	78	93	109	93	46	62	62	6015	3185	3038	5640	**2870** (ms)
6	468	1060	219	250	125	78	78	78	94	109	93	47	78	63	6092	3158	3097	5674	**2858** (ms)
7	483	1263	234	266	140	109	78	78	94	109	109	63	63	63	6049	3102	3103	5501	**3170** (ms)
8	468	1061	203	250	110	94	78	78	93	109	93	46	62	47	6079	3119	3045	5588	**2810** (ms)
9	483	1092	203	250	124	78	78	78	94	109	94	47	78	62	6063	3130	3044	5647	**2888** (ms)
10	468	1061	219	249	124	78	78	78	94	110	96	47	62	62	6078	3131	3028	5606	**2844** (ms)
11	468	1045	219	265	124	94	63	78	109	109	94	47	62	62	6116	3148	3068	5741	**2857** (ms)
12	468	1061	218	250	125	93	78	78	94	110	94	63	62	47	6098	3158	3286	5504	**2859** (ms)
13	483	1060	218	249	125	78	78	78	94	125	94	63	78	63	6041	3166	3067	5540	**2904** (ms)
14	468	1045	218	250	125	93	78	78	94	109	94	63	63	62	6061	3097	3068	5390	**2858** (ms)
15	468	1045	218	250	125	94	63	78	78	109	93	46	62	62	6109	3199	3034	5827	**2809** (ms)
16	452	1045	218	265	125	94	78	78	94	109	93	62	78	62	6129	3132	3073	5573	**2871** (ms)
17	468	1061	218	250	125	93	78	78	94	109	93	63	78	63	6005	3259	3057	5630	**2889** (ms)
18	452	1076	218	249	125	78	78	78	93	109	93	47	62	47	6042	3106	3071	5540	**2823** (ms)
19	468	1061	219	265	124	94	78	78	94	109	94	62	78	47	6100	3165	3050	5735	**2889** (ms)
20	468	1061	203	249	109	94	78	78	93	109	94	47	63	47	6125	3250	3057	5599	**2811** (ms)
21	468	1061	218	249	125	78	78	78	93	109	93	62	62	47	6066	3139	3025	5482	**2839** (ms)
22	499	1061	219	250	125	94	78	62	93	109	94	62	78	62	6150	3355	3050	5582	**2904** (ms)
23	453	1061	203	250	125	94	78	78	93	109	94	62	78	47	6021	3089	3077	5852	**2843** (ms)
24	452	1061	202	249	125	94	78	78	94	109	93	46	78	62	6240	3158	3045	5546	**2839** (ms)
25	499	1046	218	265	125	93	62	78	94	109	94	47	63	47	6033	3301	3063	5614	**2858** (ms)
26	468	1045	218	249	125	78	78	78	93	109	93	62	63	47	6064	3165	3101	5518	**2824** (ms)
27	452	1045	218	265	125	94	78	78	93	109	109	62	78	62	6090	3173	3046	5538	**2886** (ms)
28	468	1060	218	250	125	93	62	78	94	109	93	63	78	63	6045	3169	3037	5790	**2872** (ms)
29	499	1061	203	249	109	94	78	78	94	109	93	46	62	62	6036	3142	3050	5442	**2855** (ms)
30	499	1045	218	265	125	78	78	78	93	109	93	47	62	47	6015	3092	3042	5441	**2855** (ms)
31	468	1045	203	250	125	78	78	78	93	109	94	47	78	62	6163	3699	3034	5528	**2826** (ms)
32	453	1045	219	249	124	94	78	78	94	109	93	47	63	47	6015	3160	3068	5662	**2811** (ms)
33	468	1045	218	265	125	78	78	78	93	125	93	46	62	46	6070	3137	3053	5449	**2838** (ms)
34	483	1061	218	250	110	94	78	78	93	109	78	62	62	62	6296	3128	3057	5591	**2856** (ms)
35	468	1061	219	250	125	94	63	78	94	109	94	47	62	47	6081	3193	3070	5449	**2829** (ms)
36	468	1060	218	250	125	93	62	78	93	109	93	47	62	47	6127	3149	3069	5684	**2823** (ms)
37	468	1061	203	250	125	93	78	78	94	110	94	47	63	47	6041	3196	3045	5621	**2829** (ms)
38	500	1045	218	249	125	94	63	78	93	125	93	47	62	63	6059	3131	3067	5547	**2873** (ms)
39	468	1045	219	266	125	78	78	63	94	109	93	63	78	63	6036	3215	3061	5519	**2860** (ms)
40	468	1061	218	250	125	78	78	78	93	109	93	46	62	46	6206	3088	3026	5586	**2823** (ms)
41	499	1045	218	250	125	93	62	78	94	110	94	47	78	63	6174	3131	3064	5557	**2874** (ms)
42	453	1045	218	265	125	93	62	78	93	125	93	47	62	47	6030	3100	3020	5728	**2824** (ms)
43	452	1061	218	250	125	93	78	78	94	110	94	63	78	63	5981	3229	3074	6161	**2875** (ms)
44	468	1061	218	249	125	79	78	78	94	124	94	47	63	47	6060	3134	3063	5565	**2843** (ms)
45	452	1061	219	250	124	78	78	78	94	125	94	47	62	47	6117	3193	3036	5647	**2827** (ms)
46	452	1046	218	250	125	93	62	78	94	109	78	47	63	47	6108	3144	3054	5875	**2780** (ms)
47	468	1061	203	250	110	94	78	78	93	109	94	47	63	47	6112	3162	3015	5556	**2813** (ms)
48	483	1060	219	249	109	94	78	78	93	109	93	62	78	62	6042	3090	3065	5547	**2885** (ms)
49	453	1061	203	249	125	78	78	78	93	110	94	47	62	47	5992	3176	3059	5476	**2796** (ms)
50	468	1045	203	250	110	94	78	78	109	109	94	47	62	62	6291	3151	3070	5508	**2827** (ms)
AVG	**470**	**1060**	**215**	**254**	**123**	**89**	**75**	**77**	**94**	**111**	**94**	**53**	**67**	**55**	**6086**	**3174**	**3061**	**5599**	**2855** (ms)
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(µs)	(µs)	(µs)	(µs)	

Table K

SET3 Blocksize 65536B Heuristics 128 ops

Operation no. / Measurement no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Total
1	94	171	93	94	109	78	93	109	93	78	109	109	125	109	109	**1573** (ms)
2	109	187	78	78	93	78	78	94	94	78	93	93	109	109	93	**1464** (ms)
3	94	156	78	78	109	93	78	94	94	93	93	93	109	109	109	**1480** (ms)
4	109	172	109	78	109	94	78	93	93	94	94	94	109	93	109	**1528** (ms)
5	94	187	93	78	94	94	93	93	93	94	94	94	109	109	109	**1528** (ms)
6	109	172	78	78	125	94	109	110	94	94	140	125	125	109	110	**1672** (ms)
7	109	187	78	78	109	94	78	141	94	93	109	93	109	109	125	**1606** (ms)
8	109	188	94	94	109	78	93	109	94	78	109	93	109	109	93	**1559** (ms)
9	94	156	93	78	110	94	94	94	94	93	93	93	109	94	125	**1514** (ms)
10	110	172	78	78	110	94	93	109	93	78	94	109	109	110	94	**1531** (ms)
11	109	187	94	78	109	93	93	109	93	78	109	109	125	110	110	**1606** (ms)
12	109	187	78	78	109	78	94	109	93	78	109	109	125	94	109	**1559** (ms)
13	94	172	78	78	109	93	78	109	78	94	124	93	109	110	110	**1529** (ms)
14	94	171	93	78	109	94	93	93	93	78	109	109	125	94	109	**1542** (ms)
15	109	156	78	78	109	93	93	93	94	94	109	94	109	109	110	**1528** (ms)
16	94	172	94	78	109	78	94	94	78	78	109	110	125	93	109	**1515** (ms)
17	109	187	78	78	110	94	93	125	94	94	94	94	110	94	125	**1579** (ms)
18	94	156	94	78	109	93	94	94	78	94	109	94	110	109	109	**1515** (ms)
19	109	172	94	78	109	93	94	94	94	93	93	109	125	94	125	**1576** (ms)
20	109	171	110	94	109	78	94	94	78	93	109	109	109	94	94	**1545** (ms)
21	109	187	78	78	109	93	78	94	94	78	110	110	125	93	109	**1545** (ms)
22	109	187	78	78	110	78	94	94	94	94	110	109	125	109	109	**1578** (ms)
23	109	172	94	93	109	78	94	110	94	78	109	109	109	110	94	**1562** (ms)
24	109	171	94	93	109	78	94	94	94	93	109	94	109	109	109	**1559** (ms)
25	109	187	94	78	93	93	94	94	78	78	109	109	125	94	109	**1544** (ms)
26	94	172	78	78	94	94	93	93	93	94	109	94	109	93	109	**1497** (ms)
27	93	172	94	78	93	93	94	94	94	94	109	94	110	94	94	**1500** (ms)
28	110	187	78	78	110	94	78	109	78	78	109	109	109	94	94	**1515** (ms)
29	109	171	94	94	109	94	93	125	94	94	109	94	110	94	124	**1608** (ms)
30	110	172	78	78	109	78	93	109	94	78	109	109	124	109	93	**1543** (ms)
31	109	187	93	78	109	78	94	109	78	93	109	93	109	109	109	**1557** (ms)
32	109	188	94	78	110	78	94	109	78	94	110	94	109	109	109	**1563** (ms)
33	109	187	94	78	93	93	94	109	93	93	109	93	109	109	125	**1588** (ms)
34	109	172	94	78	93	93	94	94	94	93	93	93	109	109	109	**1527** (ms)
35	94	156	78	78	109	78	78	109	93	78	109	109	109	94	109	**1481** (ms)
36	109	172	94	78	109	93	94	94	78	78	109	109	110	109	93	**1529** (ms)
37	109	187	93	78	109	94	93	93	93	78	125	109	125	94	109	**1589** (ms)
38	109	172	94	78	109	93	93	109	93	94	110	94	109	109	94	**1560** (ms)
39	110	172	78	78	109	78	78	94	94	78	93	109	124	93	109	**1497** (ms)
40	94	188	94	94	110	78	94	109	78	94	109	94	110	110	94	**1550** (ms)
41	109	171	93	78	110	78	78	109	93	78	93	109	110	109	93	**1511** (ms)
42	110	172	78	78	109	93	78	94	94	93	109	93	109	94	109	**1513** (ms)
43	109	187	94	78	109	78	94	93	93	94	94	94	109	93	109	**1528** (ms)
44	109	187	78	78	94	94	93	93	78	78	110	110	110	109	93	**1514** (ms)
45	94	187	93	78	109	78	93	109	94	78	110	110	125	93	93	**1544** (ms)
46	109	171	78	78	109	93	78	109	94	78	94	94	110	109	93	**1497** (ms)
47	109	202	109	93	109	78	94	140	94	93	109	93	109	94	125	**1651** (ms)
48	110	172	109	78	109	94	78	94	94	78	93	125	125	109	124	**1592** (ms)
49	109	187	78	78	94	94	93	125	94	78	109	109	109	110	125	**1592** (ms)
50	109	171	328	359	390	78	78	125	94	93	109	109	110	110	125	**2388** (ms)
AVG	**105** (ms)	**177** (ms)	**93** (ms)	**86** (ms)	**112** (ms)	**87** (ms)	**90** (ms)	**104** (ms)	**90** (ms)	**86** (ms)	**106** (ms)	**102** (ms)	**114** (ms)	**103** (ms)	**108** (ms)	**1563** (ms)

Table L

SET1 Blocksize 65536B Heuristics 8192 ops

Operation no. / Measurement no.	1	2	3	4	5	6	7	8	9	10	11	12	13	Total
1	7207	1653	1622	3464	1186	1436	2044	796	765	7067	904	468	9553	**28622** (ms)
2	5991	1622	1622	3464	1202	1436	2044	811	748	7270	905	468	9200	**27592** (ms)
3	6053	1622	1638	3619	1232	1436	2028	812	749	7207	874	468	9595	**27748** (ms)
4	5960	1622	1622	3478	1217	1436	2044	811	764	7083	936	452	9570	**27435** (ms)
5	5975	1638	1669	3620	1186	1451	2043	811	749	6942	936	483	9216	**27512** (ms)
6	5943	1638	1638	3494	1202	1435	2043	811	765	7161	936	484	9285	**27559** (ms)
7	6115	1623	1638	3557	1217	1435	2060	827	749	7410	905	453	9323	**27998** (ms)
8	6007	1607	1623	3369	1185	1419	2028	811	749	7535	968	452	9311	**27762** (ms)
9	6037	1638	1623	3541	1185	1435	2013	811	764	6958	936	468	9169	**27418** (ms)
10	5959	1638	1623	3526	1248	1466	2028	811	811	7145	952	468	9130	**27684** (ms)
11	5990	1638	1638	3572	1248	1451	2012	812	733	7690	951	468	9669	**28213** (ms)
12	5991	1607	1607	3526	1202	1435	2059	812	749	7145	951	483	9424	**27576** (ms)
13	5975	1623	1622	3603	1186	1435	2028	811	764	7644	905	468	9287	**28073** (ms)
14	5959	1622	1607	3447	1185	1435	2044	796	749	7254	905	453	9163	**27465** (ms)
15	6006	1622	1622	3541	1185	1435	2043	811	749	7161	936	452	9241	**27572** (ms)
16	6007	1623	1622	3417	1201	1420	2044	811	749	7411	952	453	9607	**27720** (ms)
17	6006	1638	1622	3557	1201	1451	2106	811	749	7192	967	468	9238	**27777** (ms)
18	5975	1638	1607	3510	1201	1482	2043	827	749	7192	905	468	9382	**27606** (ms)
19	5991	1607	1622	3603	1201	1436	2028	811	749	7176	905	468	9510	**27607** (ms)
20	5991	1622	1622	3386	1202	1435	2043	811	749	6989	936	453	9159	**27248** (ms)
21	6037	1638	1622	3557	1186	1451	2028	826	733	7005	936	468	9343	**27496** (ms)
22	5975	1638	1638	3619	1217	1435	2074	811	765	7192	920	468	9249	**27761** (ms)
23	6006	1623	1606	3572	1217	1435	2043	811	749	7067	936	452	9147	**27526** (ms)
24	5959	1623	1638	3479	1201	1435	2028	843	749	7192	952	468	9180	**27576** (ms)
25	6021	1638	1607	3634	1201	1451	2044	811	764	6910	952	468	9164	**27510** (ms)
26	6022	1607	1622	3494	1201	1451	2060	811	764	7411	967	483	9695	**27903** (ms)
27	5960	1623	1638	3572	1202	1435	2028	827	733	7177	889	468	9459	**27561** (ms)
28	6006	1622	1623	3369	1186	1435	2043	811	749	7286	952	452	9141	**27543** (ms)
29	6006	1638	1638	3557	1217	1420	2044	811	764	7628	952	468	9599	**28153** (ms)
30	6100	1623	1622	3573	1216	1435	2028	811	749	7441	936	531	9282	**28074** (ms)
31	5960	1638	1622	3448	1217	1451	2028	812	749	7348	858	468	9341	**27608** (ms)
32	5990	1623	1623	3603	1201	1451	2013	811	749	7176	889	468	9389	**27606** (ms)
33	5991	1622	1638	3650	1216	1419	2059	827	748	7239	921	468	9232	**27807** (ms)
34	5975	1638	1638	3572	1202	1451	2013	811	748	7316	936	468	9347	**27777** (ms)
35	6006	1638	1622	3557	1202	1420	2059	811	733	7145	921	484	9189	**27607** (ms)
36	6006	1638	1638	3588	1233	1435	2059	812	749	7067	890	468	9503	**27593** (ms)
37	5991	1638	1623	3494	1185	1435	2043	811	749	7332	952	468	9232	**27730** (ms)
38	5990	1638	1607	3542	1201	1435	2044	811	764	7441	951	468	9369	**27901** (ms)
39	5990	1607	1606	3510	1201	1451	2044	827	764	7161	905	484	9221	**27559** (ms)
40	5975	1622	1623	3557	1202	1435	2012	811	749	7192	873	483	9155	**27543** (ms)
41	6006	1622	1622	3416	1202	1467	2044	811	749	7223	936	452	9234	**27559** (ms)
42	6053	1607	1669	3542	1217	1435	2028	811	764	7847	951	484	9211	**28417** (ms)
43	5990	1622	1638	3526	1201	1435	2059	811	733	7863	920	483	9241	**28290** (ms)
44	5990	1638	1622	3572	1185	1435	2028	827	765	7145	890	468	9213	**27574** (ms)
45	6084	1622	1638	3447	1233	1436	2059	826	765	7223	921	468	9308	**27731** (ms)
46	6006	1638	1622	3572	1201	1435	2028	811	749	7738	952	468	9274	**28229** (ms)
47	6021	1622	1622	3510	1201	1435	2059	812	733	7286	936	483	9206	**27729** (ms)
48	6022	1638	1638	3589	1201	1450	2012	811	749	7082	905	468	9238	**27574** (ms)
49	5975	1623	1623	3463	1201	1436	2028	811	764	7145	921	468	9176	**27467** (ms)
50	6021	1638	1623	3557	1232	1435	2029	827	748	7394	952	468	9119	**27933** (ms)
AVG	**6025**	**1628**	**1626**	**3529**	**1205**	**1439**	**2040**	**814**	**753**	**7266**	**927**	**469**	**9310**	**27730** (ms)
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(µs)	

Table M

SET2 Blocksize 65536B Heuristics 8192 ops

Operation no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	Total
Measurement no.																			
1	499	31481	218	249	172	125	78	78	93	109	93	47	78	63	6174	3103	3041	5675	**33401** (ms)
2	468	31469	218	265	125	110	78	78	93	109	93	47	62	47	6106	3137	3040	5440	**33280** (ms)
3	499	31591	218	250	109	110	78	78	93	109	94	47	63	62	6311	3083	3067	5411	**33419** (ms)
4	453	31746	203	250	125	109	63	78	94	125	94	47	63	47	6164	3146	3005	5450	**33515** (ms)
5	452	31497	203	250	125	109	78	78	93	109	93	46	62	46	6494	3285	3041	5447	**33259** (ms)
6	499	31918	202	249	109	110	78	78	93	109	94	62	62	46	6168	3071	3042	5430	**33727** (ms)
7	452	31652	218	249	125	125	78	78	93	109	94	62	78	62	6117	3142	3053	5388	**33493** (ms)
8	452	31699	218	265	125	110	78	78	93	109	94	47	78	62	6590	3295	3021	5347	**33526** (ms)
9	468	31652	218	250	125	109	63	78	94	125	94	47	62	62	6774	3083	3052	5415	**33465** (ms)
10	468	31544	219	265	124	109	78	78	94	109	94	47	63	47	6154	3043	3038	5453	**33357** (ms)
11	468	31668	218	249	125	110	78	78	94	109	94	47	78	63	6144	3107	3015	5441	**33497** (ms)
12	452	31699	218	265	125	109	78	78	94	110	94	63	62	63	6005	3152	3046	5443	**33528** (ms)
13	468	31795	219	249	124	109	78	78	94	109	93	62	62	46	6184	3142	3488	5490	**33604** (ms)
14	452	31917	203	250	109	109	63	78	93	109	93	46	78	62	6185	3066	3024	5407	**33680** (ms)
15	468	31794	203	250	125	109	78	78	93	109	93	46	62	46	6140	3065	3058	5288	**33572** (ms)
16	468	31606	219	250	125	109	78	78	93	125	93	46	62	62	6139	3160	3048	5368	**33432** (ms)
17	468	31669	219	266	125	93	78	78	93	109	93	62	62	47	6332	3085	3093	5401	**33480** (ms)
18	468	31637	219	249	124	109	78	63	94	109	94	47	63	47	6038	3103	3085	5363	**33419** (ms)
19	453	31543	218	265	125	109	78	78	94	110	94	47	63	47	6191	3107	3027	5386	**33342** (ms)
20	468	31668	218	265	125	124	78	78	94	110	94	47	62	47	6037	3047	3029	5546	**33496** (ms)
21	468	31730	218	265	125	110	78	78	94	110	94	47	62	47	6035	3151	3030	5404	**33544** (ms)
22	483	31715	203	250	125	109	78	78	93	125	93	46	62	46	6163	3057	3090	5390	**33524** (ms)
23	468	31544	218	265	125	110	78	78	78	109	93	47	62	47	6183	3149	3082	5487	**33340** (ms)
24	468	31699	203	250	125	109	63	78	94	109	94	47	78	62	6006	3085	3680	5519	**33497** (ms)
25	500	31512	219	266	125	109	78	78	93	109	78	62	62	62	6013	3156	3041	5351	**33371** (ms)
26	468	31793	218	265	125	110	78	78	94	109	93	63	63	63	5991	3058	3028	5689	**33638** (ms)
27	468	31777	202	249	109	110	78	78	109	124	94	47	63	62	6664	3158	3038	5582	**33588** (ms)
28	468	31653	219	266	125	109	78	78	94	109	94	47	63	47	6029	3125	3050	5377	**33468** (ms)
29	468	31778	218	249	109	109	78	78	94	110	94	63	63	62	6290	3115	3057	5454	**33591** (ms)
30	452	31668	219	266	125	109	78	62	93	109	94	62	78	62	6244	3182	3036	5764	**33495** (ms)
31	453	31778	218	250	125	125	78	78	94	124	94	47	63	47	6116	3063	3104	5400	**33592** (ms)
32	499	31746	218	249	125	110	78	78	94	109	93	63	78	63	6116	3087	3047	5392	**33621** (ms)
33	468	31778	219	249	125	109	78	78	109	125	93	46	62	46	6087	3138	3101	5325	**33603** (ms)
34	468	31699	219	249	109	109	78	78	94	109	93	46	62	62	6070	3158	3143	6305	**33494** (ms)
35	468	31575	203	250	110	109	78	78	94	110	94	63	78	63	6251	3060	3042	5385	**33391** (ms)
36	468	31825	218	265	125	109	78	78	93	109	94	47	78	62	6275	3138	3030	5457	**33667** (ms)
37	453	31871	203	250	110	109	78	78	109	109	93	47	62	63	6190	3111	3031	5558	**33653** (ms)
38	468	32012	219	250	125	109	78	78	94	125	94	47	62	47	6146	3103	3071	5564	**33826** (ms)
39	468	31731	219	265	124	109	78	78	94	109	78	47	63	62	6285	3139	3037	5593	**33543** (ms)
40	468	31637	218	249	125	110	78	63	94	109	94	47	63	47	6089	3072	3017	5635	**33420** (ms)
41	468	31855	218	265	125	110	78	78	94	109	94	63	63	63	6549	3146	3065	5362	**33701** (ms)
42	468	31684	218	265	125	109	78	78	94	109	93	63	63	63	6003	3075	3026	5472	**33528** (ms)
43	499	31653	218	250	125	110	78	78	94	110	94	63	78	63	6142	3144	3049	5408	**33531** (ms)
44	484	31762	203	249	124	109	78	78	94	109	93	62	63	47	6144	3137	3022	5628	**33573** (ms)
45	468	31450	219	250	125	109	78	78	94	110	94	47	63	63	6023	3077	3053	5413	**33266** (ms)
46	483	31668	219	266	125	109	78	78	93	109	94	62	78	62	6325	4064	3061	5362	**33543** (ms)
47	468	31606	218	249	109	110	62	78	109	109	94	47	78	62	6172	3145	3039	5325	**33417** (ms)
48	468	32074	203	249	124	109	62	78	78	110	94	47	63	47	6269	3132	3055	5567	**33824** (ms)
49	499	31700	203	250	125	109	78	78	94	110	78	47	62	47	6109	3135	3051	5349	**33498** (ms)
50	468	31856	203	250	140	109	78	78	93	109	94	47	78	62	6167	3130	3089	5317	**33683** (ms)
AVG	**470**	**31701**	**214**	**255**	**123**	**110**	**76**	**77**	**94**	**112**	**93**	**52**	**67**	**56**	**6192**	**3138**	**3072**	**5468**	**33518** (ms)
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(µs)	(µs)	(µs)	(µs)	

Table N

SET3 Blocksize 65536B Heuristics 8192 ops

Operation no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Total
Measurement no.																
1	156	296	125	125	156	546	546	562	531	515	8440	8393	8439	8471	8377	**45678** (ms)
2	109	250	94	94	125	484	468	546	608	546	8549	7301	7317	7347	7301	**41139** (ms)
3	109	249	109	109	125	499	499	515	484	468	7987	7971	7956	7972	7957	**43009** (ms)
4	109	249	93	93	125	484	484	499	452	452	7457	7472	7472	7473	7442	**40356** (ms)
5	109	250	94	94	125	499	499	499	468	468	7706	7691	7707	7738	7707	**41654** (ms)
6	109	249	93	93	124	483	484	483	452	452	7301	7363	7316	7317	7301	**39620** (ms)
7	109	250	110	110	124	499	500	515	483	484	7956	7956	7925	7940	7925	**42886** (ms)
8	110	250	94	109	140	468	452	468	437	437	6989	7020	7005	7036	6974	**37989** (ms)
9	109	249	93	109	140	546	514	561	499	499	8471	8471	8502	8517	8439	**45719** (ms)
10	110	250	109	109	124	484	468	483	452	437	7161	7192	7207	7176	7145	**38907** (ms)
11	109	265	109	109	125	468	468	484	437	437	7129	7083	7098	7082	7114	**38517** (ms)
12	109	250	109	109	125	484	468	483	453	437	7332	7363	7317	7348	7332	**39719** (ms)
13	93	250	109	109	125	468	468	484	453	452	7301	7348	7332	7269	7301	**39562** (ms)
14	110	250	94	94	125	484	468	484	452	453	7301	7316	7332	7254	7301	**39518** (ms)
15	110	250	109	109	124	484	468	483	452	452	7316	7316	7316	7348	7316	**39653** (ms)
16	93	234	109	109	140	499	483	499	468	468	7488	7503	7504	7519	7472	**40588** (ms)
17	110	265	109	109	124	470	468	483	452	453	7161	7208	7161	7161	7145	**38879** (ms)
18	109	250	94	110	125	468	468	468	437	421	6957	6988	6989	6989	6958	**37831** (ms)
19	109	249	109	109	141	514	499	515	468	468	8018	8003	8003	7987	7956	**43148** (ms)
20	110	249	109	109	140	468	468	483	452	437	7222	7239	7223	7223	7192	**39124** (ms)
21	109	250	94	110	125	483	484	484	453	452	7317	7348	7317	7332	7285	**39643** (ms)
22	109	234	94	110	141	499	483	499	468	453	7457	7456	7457	7456	7457	**40373** (ms)
23	109	250	109	109	125	484	468	483	437	452	7317	7317	7332	7317	7301	**39610** (ms)
24	109	249	93	109	124	483	468	483	452	452	7254	7300	7254	7286	7254	**39370** (ms)
25	109	249	109	109	125	484	468	484	453	453	7363	7394	7394	7378	7363	**39935** (ms)
26	109	234	109	109	140	484	468	484	452	436	7301	7317	7317	7316	7285	**39561** (ms)
27	109	265	94	109	125	468	468	483	436	437	7176	7208	7192	7176	7160	**38906** (ms)
28	93	234	109	109	125	531	531	531	499	499	8596	8533	8595	8549	8549	**46083** (ms)
29	94	125	94	437	124	811	515	499	499	499	8439	8362	8424	8409	8346	**45677** (ms)
30	109	250	94	109	141	546	515	546	500	484	8440	8487	8487	8486	8409	**45603** (ms)
31	110	250	94	94	125	484	483	483	452	452	7394	7394	7379	7363	7332	**39889** (ms)
32	109	249	93	109	140	484	452	483	437	437	7207	7238	7239	7238	7207	**39122** (ms)
33	109	250	94	110	125	514	499	515	484	484	8003	8035	8018	8003	8018	**43261** (ms)
34	109	265	94	93	125	484	483	499	452	452	7285	7316	7317	7270	7270	**39514** (ms)
35	109	250	94	94	141	531	515	546	499	499	8439	8440	8424	8455	8408	**45444** (ms)
36	109	265	109	109	125	484	483	499	437	437	7285	7301	7300	7285	7255	**39483** (ms)
37	109	250	94	109	141	500	484	500	452	452	7503	7551	7504	7504	7488	**40641** (ms)
38	109	249	93	109	140	530	515	546	500	500	8440	8440	8455	8486	8377	**45489** (ms)
39	109	265	94	94	125	484	468	484	437	453	7160	7160	7160	7098	7098	**38689** (ms)
40	110	265	93	109	125	500	499	514	484	468	7987	8050	8018	8002	8003	**43227** (ms)
41	109	249	109	109	125	484	468	499	452	436	7114	7130	7145	7145	7098	**38672** (ms)
42	109	250	94	110	141	483	468	499	452	452	7410	7441	7441	7442	7410	**40202** (ms)
43	109	249	93	109	125	468	468	484	452	436	7285	7300	7285	7301	7269	**39433** (ms)
44	109	234	93	109	140	515	515	531	483	468	8221	8299	8205	8221	8159	**44302** (ms)
45	109	250	94	109	125	484	468	484	452	453	7208	7238	7223	7223	7192	**39112** (ms)
46	110	250	94	109	141	546	531	546	500	500	8455	8471	8471	8455	8455	**45634** (ms)
47	110	250	109	109	124	499	500	515	468	468	7972	7972	7972	7956	7971	**42995** (ms)
48	110	250	109	109	140	484	468	484	453	452	7176	7191	7176	7192	7145	**38939** (ms)
49	125	281	94	109	124	530	514	531	499	499	8455	8424	8471	8439	8393	**45488** (ms)
50	109	249	109	109	125	499	499	531	484	468	7987	8035	8003	7972	8003	**43182** (ms)
AVG	**109**	**249**	**100**	**113**	**130**	**501**	**486**	**503**	**467**	**462**	**7638**	**7627**	**7622**	**7618**	**7592**	**41220** (ms)
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	

Table O

File Size: 5 MB

DB size before Measurement No.	10	50	100	200	
1	2013	1981	2066	2019	(ms)
2	2012	1981	2019	2003	(ms)
3	2030	2028	2019	2003	(ms)
4	2028	2028	1973	2050	(ms)
5	2028	1982	2034	2003	(ms)
6	2028	2028	2019	2035	(ms)
7	2043	1996	2004	2035	(ms)
8	2059	1997	2003	1972	(ms)
9	2043	1997	2050	2003	(ms)
10	2028	1997	1987	2019	(ms)
11	2028	1997	2034	2019	(ms)
12	2044	1981	1972	2035	(ms)
13	2013	1981	2035	2050	(ms)
14	2028	1997	1988	1988	(ms)
15	2059	1981	2019	2035	(ms)
16	2028	2012	1987	2003	(ms)
17	2028	2012	1988	2034	(ms)
18	2028	1997	1988	2004	(ms)
19	2028	1981	2034	2050	(ms)
20	2029	1996	1988	1987	(ms)
21	2044	1997	2034	2035	(ms)
22	2012	2012	1972	2050	(ms)
23	2028	1996	1972	2003	(ms)
24	2028	1997	1988	2050	(ms)
25	2044	1997	2003	2003	(ms)
26	2060	1997	1987	2034	(ms)
27	2044	2012	1988	1988	(ms)
28	2028	1997	2019	2050	(ms)
29	2043	1997	1988	2003	(ms)
30	2013	1997	1987	2035	(ms)
31	2028	2028	1987	2019	(ms)
32	2013	2028	1988	2019	(ms)
33	2028	2012	1988	2019	(ms)
34	2013	2028	1988	2034	(ms)
35	2012	1981	1972	2003	(ms)
36	2028	2028	2034	2066	(ms)
37	2012	1996	2019	1988	(ms)
38	2028	2012	1972	2034	(ms)
39	2013	1982	2003	2018	(ms)
40	2013	2013	1988	2035	(ms)
41	2028	1997	2004	1988	(ms)
42	2044	1997	2003	2019	(ms)
43	2012	2012	1972	2034	(ms)
44	2044	2012	1987	2050	(ms)
45	2028	1997	2050	2003	(ms)
46	2013	1981	2019	2035	(ms)
47	2012	1996	1972	2003	(ms)
48	2028	1997	1987	2034	(ms)
49	2028	2012	2019	2034	(ms)
50	2028	1997	1972	2066	(ms)
AVG	**2028**	**2001**	**2001**	**2022**	
	(ms)	(ms)	(ms)	(ms)	

Table P

DB Size before: 10 MB

File Size	1 MB	5 MB	10 MB	20 MB	
Measurement No.					
1	460	2013	4305	8096	(ms)
2	460	2012	3931	8049	(ms)
3	461	2030	3962	8096	(ms)
4	476	2028	3954	8127	(ms)
5	460	2028	3930	8112	(ms)
6	445	2028	3868	8066	(ms)
7	461	2043	3946	8113	(ms)
8	461	2059	3915	7910	(ms)
9	461	2043	3900	8018	(ms)
10	461	2028	3899	8034	(ms)
11	461	2028	3930	8019	(ms)
12	461	2044	3899	8049	(ms)
13	476	2013	3915	8081	(ms)
14	461	2028	3931	8205	(ms)
15	460	2059	3915	8034	(ms)
16	445	2028	3899	8019	(ms)
17	476	2028	3900	8049	(ms)
18	444	2028	3931	8065	(ms)
19	461	2028	3899	8050	(ms)
20	445	2029	3931	7941	(ms)
21	476	2044	3962	7972	(ms)
22	461	2012	3992	8018	(ms)
23	476	2028	3931	7987	(ms)
24	460	2028	3977	8003	(ms)
25	476	2044	3931	8034	(ms)
26	460	2060	3930	8112	(ms)
27	461	2044	3899	8034	(ms)
28	445	2028	3915	8019	(ms)
29	461	2043	3933	8034	(ms)
30	460	2013	3931	8034	(ms)
31	460	2028	3946	8081	(ms)
32	460	2013	3899	8003	(ms)
33	476	2028	3915	7972	(ms)
34	461	2013	3946	8112	(ms)
35	461	2012	3930	8018	(ms)
36	461	2028	3978	7987	(ms)
37	460	2012	3900	8081	(ms)
38	461	2028	3915	8065	(ms)
39	445	2013	3930	8065	(ms)
40	461	2013	3961	8003	(ms)
41	461	2028	4118	8050	(ms)
42	461	2044	4075	8049	(ms)
43	445	2012	4071	8081	(ms)
44	461	2044	4071	8034	(ms)
45	461	2028	4307	8003	(ms)
46	460	2013	4148	8034	(ms)
47	460	2012	4117	8096	(ms)
48	461	2028	4103	7987	(ms)
49	460	2028	4460	8019	(ms)
50	460	2028	4040	8003	(ms)
AVG	**461** (ms)	**2028** (ms)	**3980** (ms)	**8042** (ms)	

Table Q