Computational Science Technical Note **CSTN-197**

Performance of Mining Medium-to-Large-Scale Scientific Simulation Data

S. G. Morgan and K. A. Hawick

2013

Many scientific simulations generate bulky data sets that must be mined for observable features. It is often not computationally feasible to do this in real time and the data must be saved for subsequent "off line" analysis either by separate software or sometimes by direct human visualisation. We present some scoping analysis and preliminary software approaches for mining medium to large scale data sets in the form of time slices or model configurations. We report on current storage and visualisation technology response and interaction times for mining scientific simulations on regular lattices using hyper-bricks of model configurations.

Keywords: big data; data mining; data visualization; scientific simulations

BiBTeX reference:

```
@INPROCEEDINGS{CSTN-197,
        author = {S. G. Morgan and K. A. Hawick},
        title = {Performance of Mining Medium-to-Large-Scale Scientific Simulation
                Data},
        booktitle = {Proc. 14th International Conference on Internet Computing and Big
                Data (ICOMP'13)},
        year = {2013},
        number = {CSTN-197},
        pages = {ICM4063},
        address = {Las Vegas, USA},
        month = {22-25 July},
        publisher = {WorldComp},
        institution = {Computer Science, Massey University, Auckland, New Zealand},
        keywords = {big data; data mining; data visualization; scientific simulations},
        owner = {kahawick},
        timestamp = {2013.04.22}
}
```

Technical Report CSTN-197

Performance of Mining Medium-to-Large-Scale Scientific Simulation Data

S.G. Morgan and K.A. Hawick
Computer Science, Massey University, North Shore 102-904, Auckland, New Zealand
email: scottgarrymorgan@gmail.com, k.a.hawick@massey.ac.nz
Tel: +64 9 414 0800   Fax: +64 9 441 8181

April 2013

ABSTRACT

Many scientific simulations generate bulky data sets that must be mined for observable features. It is often not computationally feasible to do this in real time and the data must be saved for subsequent "off-line" analysis either by separate software or sometimes by direct human visualisation. We present some scoping analysis and preliminary software approaches for mining medium to large scale data sets in the form of time slices or model configurations. We report on current storage and visualisation technology response and interaction times for mining scientific simulations on regular lattices using hyper-bricks of model configurations.

KEY WORDS
big data; data mining; data visualization; scientific simulations.

1 Introduction

Numerical simulations often generate relatively large data sets that need "offline" analysis. A time series analysis can generally not be conducted until the simulation sequence is completed and data-mining and visualisation tools are needed to support scientists conducting post simulation mining activities.

Computational simulations include: discrete event simulation models [12]; complex system models based on particles [16] or agents; or time-integrated field models all produce bulk data that is often stored and post processed in this manner. Often such data is spatially oriented [4] but can come in a myriad of different storage formats. In this present paper we focus on "hyper-bricks" of regular data that come from models or simulations where the key data structure is a multidimensional array or "brick" of data. The cells of such data might be simple pixels or volume element - voxels; or multi-channel data with several scalar or vector properties at each spatial location [21].

Our experience is that a semi-interactive pattern matching identification [18] of even just a visual inspection of the various "slices" of such data bricks can be very instructive as part of a numerical experiment.

Such data arises from environmental modelling [22] and geographical systems [1, 23] as well as from complex systems models in application areas like: forest fire systems [3]; lattice gases and complex fluids [10]; critical systems [8]; and ecological systems [5].

Relational or graph data [7, 20] from biological [6] and other applications [19] can also of course be mined using statistical pattern matching, but such databases tend to have their own specialist performance optimization strategies. In the present paper we focus on data that is held in simpler more "raw" formats such as rasters and hyper-bricks rather than relational and textual data.

There has been considerable interest in the literature on strategies to mine very large time series data [15] and on how to manage such data on distributed systems [13]. In this present paper we investigate practical matters concerning the manipulation of medium to large data sets that can be manipulated by post-simulation analysis software on a modern desk-top or desk-side scaled computer system.

Our article is structured as follows: In Section 2 we describe the hyperbrick data manipulation problem in detail and discuss file format and data layout issues for coping with arbitrary dimensional data. In Section 4 we report on experiments to characterise read and write performance of large contiguous bricks of such data. We discuss implications of typical read and write performance for individual and bulk data for a range of different storage devices in Section 5. We offer some conclusions and areas for further work on bulk data manipulation in Section 6.

2 Scientific Model Data Bricks

There are some sophisticated 3D solid design file formats available that are used by proprietary and some open computer aided design tools. Even for those formats that are open their complexity makes it a cumbersome burden to interface a stand-alone simulation program to them. A very simple file format family was designed to provide a bridge between the *Cubes* visualisation program and the sort of simulation code our research group regularly develops in C, C++, Java and other languages.

The hyperbrick file format – with file ending ".hbrk" – was inspired by the incredibly useful portable pixmap format family (often known as "NetPBM") designed by Poskanzer and developed by Henderson [11]. Researchers have been using ppm and pbm formats with 2D simulation programs for over two decades and their value is largely due to their simplicity. One can off the top of one's head code up C/C++/Java to generate, or read and write these formats.

In a sense therefore, the "H1" hyperbrick is a generalisation of the pgm 2D greymap image file format, for the case of 3D data.

```
H1
# a comment or header line
# another comment
1
3 64 64 32
hyper-raster-of-unsigned-chars
```

Figure 1: The .hbrk hyperbrick file format for a 3-d data set of unsigned chars with spatial extent $x = 64 \times y = 64 \times z = 32$.

Figure 1 shows the .hbrk file format, consisting of a two character textual header "H1" followed by a newline and an optional series of comment lines starting with a hash character. The subsequent integer – in this case a size of "1" denotes the number of bytes in each payload entity. If one wanted voxels to be allowed to take on $2^2 4$ different levels - like portable pixmap pixels, then one could use a size of "3" to denote three bytes per voxel. The next line gives the dimensionality d of the hyperbrick – usually $d = 3$ for examples discussed in this document, followed by exactly d integer edge lengths in order of increasing significance – so in the example shown $L_x = 64, L_y = 64, L_z = 32$. This line is terminated by a newline and the remainder of the ".hbrk" file is a set of binary characters in the "hyper-raster" order implied by the dimensionality and lengths. So in the example thei_x index would move fastest, the i_y next fastest and so forth.

Many of the simulation codes we work with use what is known as "k-indexing" whereby a single integer indexes into the d-dimensional hyperbrick and:

$$k = i_x + L_x \times i_y + (L_x \times L_y) \times i_z \tag{1}$$

These ideas are very useful for manipulating rectilinear data independent of the dimension and are described in [9] and can be generalised to specify data transforms [14].

The Netpbm format family supports both "raw" and textual formats for pix maps and grey maps. The full "hrbk" format family is still under development but the present plan is to support together data types so that for example "H4" maps to 32-bit integer data, "H8" maps to 64-bit integer data. The "H1" unsigned-char type can also be used with data cell length 4 or 8 of course to encode this as long as you do not care about byte order within the cells.

A useful variation that is easily described here is the **sparse** version or ".sbrk" sparse hyperbrick format. We found many of our programs deal with a model that can be represented as cells of one or more values in a "sea of zeroes" and consequently it is a waste of space explicitly saving all the zeros.

```
S1
# a comment or header line
# another comment
1
3 64 64 32
63455 255
67551 128
71647 192
```

Figure 2: The .sbrk sparse hyperbrick file format for a 3-d data set of unsigned chars with spatial extent $x = 64 \times y = 64 \times z = 32$.

Figure 2 shows the .hbrk file format which is similar to the "H1" format but has "S1" as its magic prefix, and embodies the assumption that the entire hyperbrick has voxel values of zero **except** the explicitly stated (k, v) pairs giving the k-index and voxel value both encoded as unsigned integers. In the example shown, the following (x, y, z) indices give rise to the k-indices:

```
(31 31 15)->k of 63,455, voxel value 255
(31 31 16)->k of 67,551, voxel value 128
(31 31 17)->k of 71,647, voxel value 192
```

The sparse hyperbrick format could also be extended into a family of formats with different magic header characters to convey type information but the simple "S1" format with the type-code "1" stating that each voxel is limited to 2^{8^1} levels suffices for most purposes.

These sort of data files with a mixed textual header and binary or raw-readable data have rather gone out of fashion at present, with a surprising number of inappropriately large XML formats being common. The simplicity of the hbrk and sbrk formats means that they can be read using built-in language I/O capabilities without recourse to needing complicated parsers.

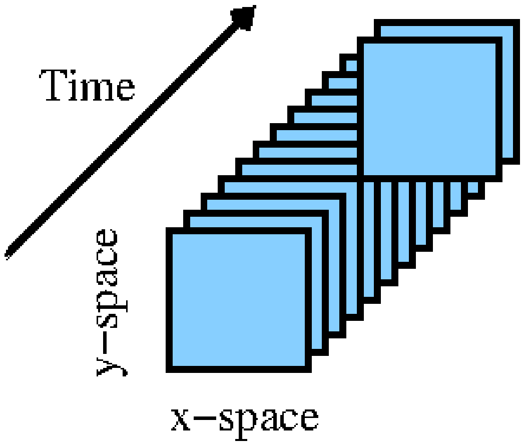

Figure 3: Time sequence of planes (which could be hyper-bricks themselves) arranged with a short window of the whole data pulled out and being analysed at once.

The k-indexing notion extends the notion of a raster contiguous image format to arbitrary dimensions. The main point is that the k-indexing allows the hyperbrick of data to be treated as a block of data that is contiguous in memory or on the storage medium. It can be read and written in a single call to a low-level read/write service call within the operating systems kernel or file systems daemon.

Figure 3 shows a time sequence of hyper-bricked data. A sub-sequence or window of the overall sequence is pulled out and analysed at once. It is therefore important to be able to read and write blocks of the overall sequence rapidly. It is therefore important to assess the current attainable performance for various storage devices.

3 Data Access Experiments

The given experiment was to test the performance on varying Hard Drive Disks and the effect of using smaller individual files in comparison to a single file totaling in equal size. A custom written program in C/C++ was used and run on the different Hard Drives and RAID formats to give full control over how the information was written and read. The physical hard drives ranges from an IDE 5200rpm based disk to a Solid State Drive. This way the variation based on physical hardware can be seen and how it can impact performance. The basis of having multiple small files being 1 Megabyte in size then totaling up to a single large file of the combined size was to assess how much the latent performance of reading and writing was affected by buffering.

Using a custom written program in C/C++ gave full control on how reading and writing were used and hence how they performed. Primarily choosing fread and fwrite as the way to read and write information to and from the hard drive, this gave the ability to be concerned with the type format the data could be. Pseudo randomly generated information was created for writing the information which was stored in a buffer then the timing was given for when the write function was called. With this implementation this helped limit any chances patterned data in which operating system could effect the timing in which random data is read or written by caching.

The test was broken down into four main parts. The initial writing of the single large file created at the set size, after the writing of the single small files. This is then followed by the reading of the same files. To make sure that no caching was done for the file so it would not effect the reading or writing time to the hard drive, commands in the terminal were used to clear the cache, for example Linux's $echo3 > /proc/sys/vm/drop_caches$. These commands where done during the test however they were not placed in when timing was done, so it will not have any effect on the timing results. The size spacing for each of the tests were done in 200MB intervals and was tested ten times each up to the limit of 2000MB.

Algorithm 1 Writing algorithm and random data generation

```
declare $buffer$[]
for all $unsigned\ characters$ in $buffer$ do
    randomise $buffer$[i] with $unsignedcharacter$
end for
create $emptyfile$
begin timing
write from buffer to file
end timing
close file
record timing
```

Algorithm 1 shows the basic constructs of how the process of writing a single large file and the generation the random data. Initially a buffer is generated and allocated the memory required for the final size of the file being created. Then the buffer is filled with randomly generated unsigned characters where the randomisation has been seeded from time using the rand function as shown in Figure 4. Once the buffer has been filled, an empty file is created in which the data from the buffer will be stored. The timing starts and will end once the writing function has ended. The function used was fwrite as shown in Figure 4. When this has completed the file is then closed and the timing is recorded and stored in a external file.

4 Performance Results

Performance between the variety of disks while reading and writing can be clearly seen. The style (using the same type of hard drive but using multiple devices with a RAID format) in which data has been stored also seems to have impacted the hard drive performance. Clear separations between the physical devices are seen as the older devices seem to be slower in all areas between both large and small files in comparison to

```
for (int i = 0; i < dataSize; i ++) {
    cp[i] = rand() % 255;
}
FILE * fp;
for (int i = 0; i < 10; i++) {
   char file_name[200];
   sprintf(file_name,"single_large_%i.data",i);
   fp = fopen( file_name, "w");
   assert( fp != NULL );
   t1 = mytimer();
   fwrite(cp, m, dataSize, fp);
   t2 = mytimer();
   t3 = t2-t1;
   timerArray[i] = t3;
}
free(cp);
fclose(fp);
```

Figure 4: Writing function and data generation

the newer technologies such as solid state drives.

The latency itself represents the time taken just for the overhead of the file without including the information based on reading or writing to and from the buffer. This would include the spin up time of the hard drive if mechanical, finding a place to put the file on the platter (or memory if a solid state drive) or anything else that would take up time that is not involved with writing or reading the information into the buffer [17]. Table 1 shows the results of all the hard drives based on information measured from reading a single file. The information that represents IDE 5400rpm, Sata 7200rpm, and RAID 1 shows a negative value essentially stating based off the line of best fit the overhead of reading a single file would be in the negative which is impossible. However all these values have a high error rate resulting in these values to become invalid. RAID 0 gave a more reliable result of stating the the initial overhead time for reading a single file resulted in the time taken to be 87ms with a standard error of 16. Most of these results seem very varied and no pattern can been seen, however the timed measured is very small compared to the total time taken for reading and writing files.

Table 2 and Table 3 represents the basic transfer speeds being recorded to and from the buffer. Table 2 shows the speed in which the files are written to the hard disk from the buffer with the randomly generated data stored in the buffer. This clearly shows that older hard drives such as the IDE 5400rpm ATA has a much slower transfer rate (being only 26.37 MB/s with a standard error of 2MB/s) than the Solid State(244MB/s with an error of 44MB/s). This same pattern can also be seen in Table 3 with the read speeds essentially mirroring its counterpart(except for the solid state which has a varying read and write speed). These results are based on just a large single file as from other data represented in Figure 9 and Figure 6 shows how being stored can effect the transfer rate. Most of the values of all the data seem to be very stable as they seem to have very minimal error rates. This shows the average rate that the hard drives were able to transfer information. The only data represented in these tables that seems unnatural is the RAID 5 read and write. This data seems to be transferring at a much higher rate than the rest of the data in comparison, so this maybe an anomaly in regards to this piece of individual data.

Figure 6 and other results gained shows how timing for both read and writing of the varying file types. This clearly shows the impact in which the way files are stored can effect performance when being read and written to the buffer. A clear definition is seen in regards to the effect of reading in multiple files to the buffer compared to reading in a single file. This seems to be far slower and takes more time on how files are read in. However reading and writing seems to be very similar, however in other results this has varied slightly with writing multiple files on average seems to be slower than writing a single file.

The RAID format clearly can increase the performance of a hard drive. This is shown with the performance increase with RAID 0 (this is being setup in which two hard drives share the load essentially increasing its performance). With two hard drives setup (SATA 3 7200rpm HDD) in RAID 0 the performance has double in comparison to just using a single hard drive as shown in Table 3. RAID 1 (mirroring the device for redundancy) showed a slight performance increase compared to the SATA 3 7200rpm single device however this is very minimal in comparison. RAID 5 format (providing increase in performance and redundancy requiring four hard drives) gave very unusual results as these systematically doubled the RAID 0 results in bandwidth through put.

Unusual anomalies have shown themselves in way different hards store information. Figure 9 shows that the time taken for it to write multiple files took longer than reading them, while all other tests represented this in the the opposite way. Figure 10 shows that writing a single file took substantially longer than all other tests taken. However as shown, the results for writing the single large file is very scattered suggesting an issue when writing as timing between the data points vary in an unusual pattern of every second point is smaller than the previous.

Device	Average Read Latency(ms)
IDE 5200rpm ATA HDD	-2926 ± 877
SATA 7200rpm HDD	-600 ± 400
RAID 0 (SATA 7200rpm)	87 ± 16
RAID 1 (SATA 7200rpm)	-471 ± 525
RAID 5 (STAT 7200rpm)	30 ± 60
Solid State HDD	421 ± 210

Table 1: Latency speeds from reading single file

Device	Average Write Speeds (MB/s)
IDE 5200rpm ATA HDD	26.37 ± 2
SATA 7200rpm HDD	132.4 ± 1.3
RAID 0 (SATA 7200rpm)	362 ± 12
RAID 1 (SATA 7200rpm)	173 ± 1.3
RAID 5 (STAT 7200rpm)	620 ± 42
Solid State HDD	244 ± 44

Table 2: Average hard drive write speeds

Device	Average Read Speeds (MB/s)
IDE 5200rpm ATA HDD	32.42 ± 1
SATA 7200rpm HDD	146.8 ± 0.5
RAID 0 (SATA 7200rpm)	349 ± 11
RAID 1 (SATA 7200rpm)	170 ± 1.1
RAID 5 (STAT 7200rpm)	692 ± 31
Solid State HDD	354 ± 2

Table 3: Average hard drive read speeds

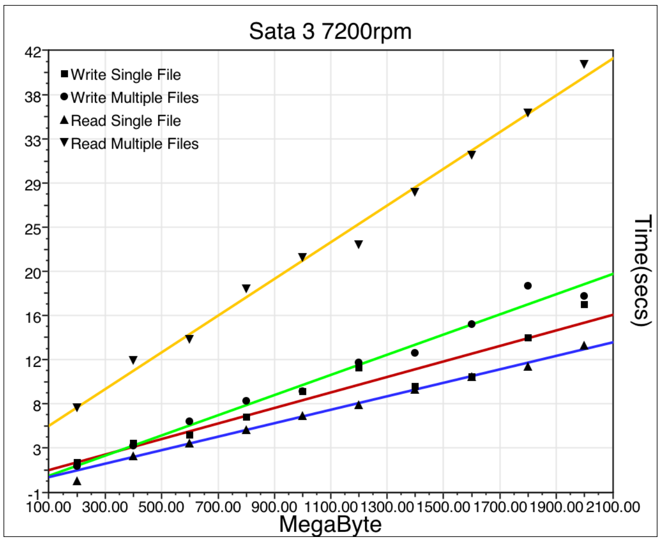

Figure 5: 7200rpm SATA 3 Drive

5 Discussion

With the results given, this shows a correlation between the way the files are stored and how being stored can effect the performance. It seems that overhead while minimal, can slowly add up to decrease performance while a file of equivalent size but stored in its entirety will have a better performance. However, issues arise with the limitation of hardware and memory to how big the file can be stored.

There are also unseen factors that can effect the performance of reading and writing and that is the current state of the disk and fragmentation. Since a hard drive needs to put information on the sectors it would be more efficient if it was able to put the information on the same track continuously rather than having the information scattered as this would increase overhead [17]. This effect however is only for mechanical hard drives as the solid state hard drives [2] do not use platters to store the information.

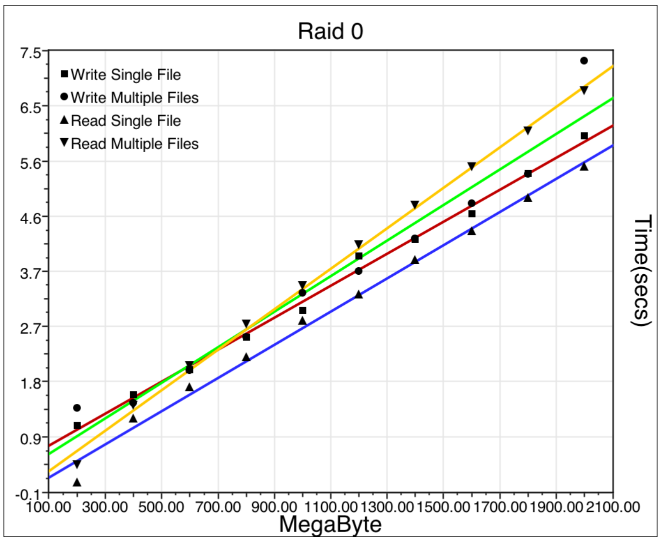

Figure 6: RAID 0 with 7200rpm SATA 3 Drives

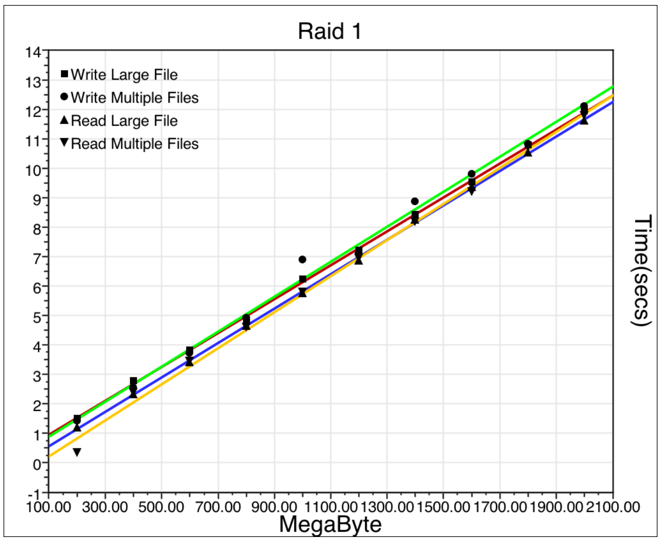

Figure 7: RAID 1 with 7200rpm SATA 3 Drives

Using a RAID format increased performance (for RAID styles that are meant to increase performance for example RAID 0 and 5) of the same type of hard drive as shown in Table 3, while the needed amount of physical hard drives increase (dependant on the type of RAID format this can vary), this shows that by using a format of RAID this greatly increases the bandwidth that it can pass through. Other RAID formats such as

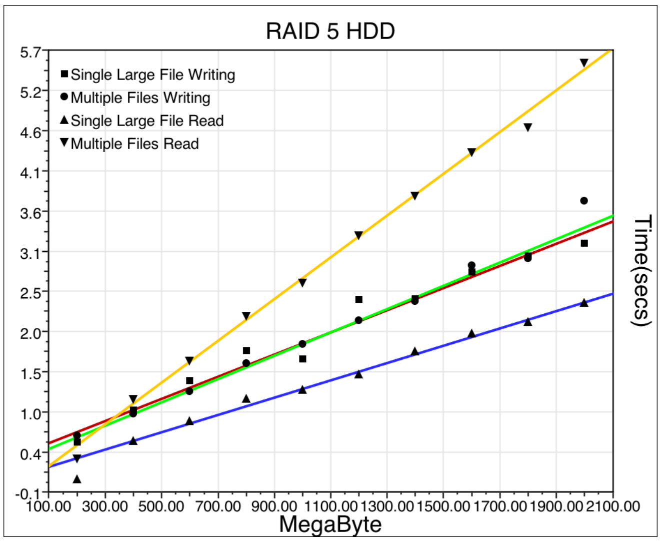

Figure 8: RAID 5 with 7200rpm SATA 3 Drives

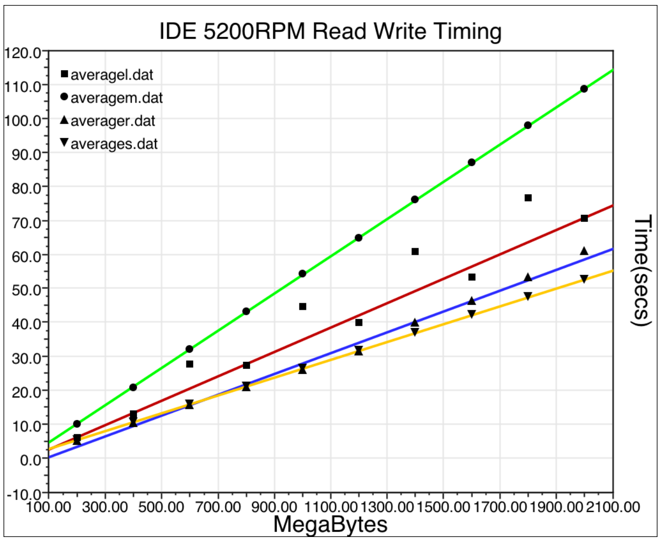

Figure 9: IDE 5200rpm ATA Drive

RAID 1 and 5 also give the ability for redundancy which could be useful incase of hard drive failure which if used for a single hard drive, this would result in loss of all data.

The variation in which the type of hard drive(Sata 3 7200rpm, IDE 5400rpm and Solid State) showed a complete variation on the way storing files effected time for the transfer. The Sata 3 7200rpm drives showed that reading multiple files has the longest time take (even in RAID 0 and 5 format) while with the IDE 5400rpm hard drive, writing multiple files seems to be significantly slower. The solid state hard drive had an issue with reading writing single files. This could be related each device and how they were designed and caching techniques.

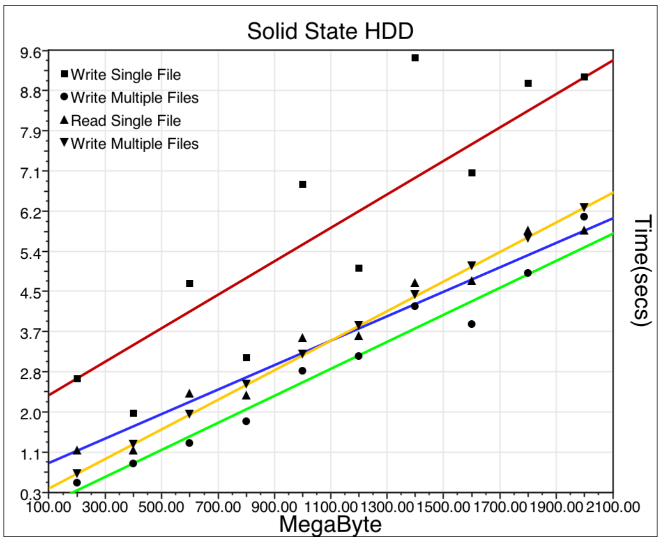

Figure 10: Solid State Hard Dive

6 Conclusion

A clear correlation has been seen in regards to how newer devices such as the solid state hard drives are out performing the older mechanical drives. This can be clearly seen with the data presented and is not unexpected. Using a RAID style format this increases the performance of the hard drives, however requires the use of more hard drives.

Storing information in a RAID format can also give the use of redundancy and bandwidth increase if using a RAID 5 format. While this will give redundancy, this will be at the sacrifice of the amount of space available per hard drive as the backup information of the data is stored incase of a failure.

The way files are stored also seem to play a major factor in the time taken of reading in files. The concept of storing many smaller files instead of a single large file seems to impact the speed in which files are written and read for mechanical hard drives. Due to the way that mechanical hard drives store information this is unsurprising that having to store multiple files will generate more overhead as having to find the individual files scattered over the platters.

Due to the nature of the solid state and the way it stores information this is not affected any mechanical parts so files are not will not be scattered and overhead is dramatically decreased with reading and writing multiple files the same as reading a single large file of equivalent size [2].

From the test given, it seems that the solid state hard drives seem to be much faster in ever aspect compared to the IDE and Sata 3 mechanical hard drives. While storing data in a RAID format using mechanical hard drives gives a very close results equivalent to a solid state, this is at the cost of using multiple hard drives.

If solid state drives are stored in a RAID format (based on the changes of bandwidth from the Sata 3 7200rpm) this would clearly increase the performance to far exceed that of the mechanical drives. However due to solid states being a far newer technology, the cost in relation to the amount of space available for the capacity of the device is higher in comparison to the current mechanical drives available.

Overall it seems that with the information given it varies on the users available equipment and the memory limitation of the computer. While It does seem with mechanical drives that storing the information as a single large file will give better speeds, this is limited to the amount of memory available to the user on the current computer they are reading it on. While devices like the solid state seem to have very minimal effect on the way the file is stored, the best choice on how the file is stored will be dependant on the physical hardware it self and what is available for the user.

In summary we have evaluated the read and write performance of several storage devices managing contiguous blocks of data suitable for simulation configuration hyperbricks. Having determined how multi-dimensional systems could be encoded in this manner we have found that desktop and desk-side devices give an adequate read and write performance for a semi-interactive data mining analysis of simulation results. In particular there is strong evidence in favour of using solid state storage devices where possible, for this sort of work.

References

[1] Coddington, P.D., m. W. Grigg, Fabbro, S.J.D., Hawick, K.A., James, S.P., Lo, E.H.S., Lut, A.K., Mason, K.D., Owen, M.J.: Interfacing to on-line geospatial imagery archives. In: Proc. 27th An. Conf. of AURISA. Leura, NSW, Australia (22-26 November 1999)

[2] Cornwell, M.: Anatomy of a solid-state drive. Communications of the ACM 55, 59–63 (2012)

[3] Gu, P., Zheng, Y.: A new data mining model for forest fire cellular automata. In: Proc. Fifth Int. Conf. on Intelligent Computation Technology and Automation (ICICTA 2012). pp. 37–40. CPS, Zhangjiajie, Hunan, China (12-14 January 2012)

[4] Guo, D., Gehagen, M.: Spatial ordering and encoding for geographic data mining and visualization. J. Intell. Inf. Sys. 27, 243–266 (2006)

[5] Hawick, K.A.: Spectral analysis of growth in spatial lotka-volterra models. In: Proc. International Conference on Modelling and Simulation. pp. 14–20. No. 685-030, IASTED, Gabarone, Botswana (6-8 September 2010)

[6] Hawick, K.A.: Applying enumerative, spectral and hybrid graph analyses to biological network data. In: Int. Conf. on Computational Intelligence and Bioinformatics (CIB 2011). pp. 89–96. IASTED, Pittsburgh, USA (7-9 November 2011)

[7] Hawick, K.A., Coddington, P.D., James, H.A., Patten, C.J.: On-line data archives. In: Proc. 34th Hawaii Int. Conf. on System Sciences. pp. 1–10. No. DHPC-021, HICSS, Hawaii, USA (January 2001)

[8] Hawick, K.A., Playne, D.P.: Modelling, Simulating and Visualizing the Cahn-Hilliard-Cook Field Equation. International Journal of Computer Aided Engineering and Technology (IJCAET) 2(1), 78–93 (2010), inderscience

[9] Hawick, K.A., Playne, D.P.: Hypercubic Storage Layout and Transforms in Arbitrary Dimensions using GPUs and CUDA. Concurrency and Computation: Practice and Experience 23(10), 1027–1050 (July 2011)

[10] Hawick, K.: Visualising multi-phase lattice gas fluid layering simulations. In: Proc. International Conference on Modeling, Simulation and Visualization Methods (MSV'11). pp. 3–9. CSREA, Las Vegas, USA (18-21 July 2011)

[11] Henderson, B.: Netpbm history. Web Site (2007), `http://netpbm.sourceforge.net/history.html`, last Visted October 2010

[12] Hossain, M., Harari, N., Semere, D., Martensson, P., Ng, A., Andersson, M.: Integrated modeling and application of standardized data schema. In: Proc. 5th Swedish Production Symposium (SPS-12). pp. 473–478. Linkoping, Sweden (6-8 November 2012)

[13] James, H.A., Hawick, K.A.: Scientific data management in a grid environment. Journal of Grid Computing 3(1-2), 39–51 (September 2005), iSSN: 1570-7873 (Paper) 1572-9814 (Online)

[14] Johnson, M.J., Hawick, K.A.: Optimising energy management of mobile computing devices,. In: Proc. Int. Conf. on Computer Design (CDES'12). pp. 1–7. WorldComp, Las Vegas, USA (16-19 July 2012)

[15] Keogh, E., Kasetty, S.: On the need for time series data mining benchmarks: A survey and empirical demonstration. Data Mining and Knowledge Discovery 7(4), 349–371 (October 2003)

[16] Mehta, S., Hazzard, K., Machiraju, R., Parthasarathy, S., Wilkins, J.: Detection and visualization of anomalous structures in molecular dynamics simulation data. In: Proc. IEEE Visualization 2004. pp. 465–472. Austin, Texas, USA (10-15 October 2004)

[17] Ng, S.W.: Advances in disk technology: Performance issues. IEEE Computer May, 75–81 (1998)

[18] Rao, C.S., Gupta, M., Murthy, K.V.S., Rajanikanth, J.: Sequetial pattern mining based on multi dimensional sequence data - a case study. Indian Journal of Computational Intelligence and Systems Sciences 1, 1–5 (2013)

[19] Sbalzarini, I.F.: Modeling and simulation of biological systems from image data. Bioessays 35, 482–490 (2013)

[20] Schietgat, L.: Graph-based data mining for biological applications. AI Communications 24, 95–96 (2011)

[21] Shekhar, S., Zhang, P., Huang, Y., Vatsaval, R.: Trends in Spatial Data Mining, chap. 3, pp. 833–851. Springer (2005), in Data Mining and Knowledge Discovery Handbook

[22] Trepos, R., Masson, V., Cordier, M.O., Gascuel-Odoux, C., Salmon-Monviola, J.: Mining simulation data by rule induction to determine critical source areas of stream water pollution by herbicides. Computers and Electronics in Agriculture 86, 75–88 (2012)

[23] Yang, H., Pathasarathy, S., Mehta, S.: A generalized framework for mining spatiotemporal patterns in scientific data. In: Proc. ACM Int. Conf. on Knowledge Discovery and Data Mining. pp. 716–721. Chicago, Illinois, USA (21-24 August 2005)