Time taken to write data in parallel on Lustre follows extreme statistics

Richard Henwood
ARM, Inc, 1 Enrico Terrace,
5707 Southwest Parkway,
Austin, Texas, USA, 78735
richard.henwood@arm.com

N. W. Watkins
Centre for the Analysis of Time Series, London School of Economics, Houghton Street, London, UK
Centre for Fusion, Space and Astrophysics, University of Warwick, UK.
Open University, Milton Keynes, UK.

S. C. Chapman
Centre for Fusion, Space and Astrophysics, University of Warwick, UK.
Department of Mathematics and Statistics, University of Tromso, Norway, Max Planck Institute for the Physics of Complex Systems, Dresden, Germany

R. McLay Texas Advanced Computer Center, 10100 Burnet Rd, Austin, Texas, USA

ABSTRACT

In both high-performance computing (HPC) environments and the public cloud, the duration of time to retrieve or save your results is simultaneously unpredictable and important to your over all resource budget. It is generally accepted ("Google: Taming the Long Latency Tail - When More Machines Equals Worse Results", Todd Hoff, highscalability.com 2012) , but without a robust explanation, that identical parallel tasks do take different durations to complete – a phenomena known as variability. This paper advances understanding of this topic. We carefully choose a model from which system-level complexity emerges that can be studied directly. We find that a generalized extreme value (GEV) model for variability naturally emerges. Using the public cloud, we find real-world observations have excellent agreement with our model. Since the GEV distribution is a limit distribution this suggests a universal property of parallel systems gated by the slowest communication element of some sort. Hence, this model is applicable to a variety of processing and IO tasks in parallel environments. These findings have important implications, ranging from idealized performance characteristics for parallel codes to detecting degraded behaviour at extreme scales.

Keywords

extreme | parallel computing | variability

1. INTRODUCTION

PDSW-DICS '16 November 14th 2016, Salt Lake City, UT

ACM ISBN 978-1-4503-2138-9.

DOI: 10.1145/1235

Where they exist at all, current models for variability on HPC systems implicitly assume I/O variability follows a normal distribution with the mean and standard deviation the only measure of interest [12, 19, 30, 22, 26]. An attempt to fit the tail of task duration to the log-normal distribution has also been made [35] with limited success. Independent of this work, there are an increasing number of phenomena in computer science and beyond that are best modeled by methods of extreme statistics [14, 3, 15, 10, 23, 11, 33, 4, 2, 7, 20, 28, 8, 24].

2. MODEL

The modern theory of extreme value distributions can be traced back to the 1920's and two mathematicians: Fisher and Tippett. They considered [13] extreme values of n samples, each of size m drawn from the same underlying population. Provided the population values are independent and identically distributed (i.i.d.), they showed that the distribution of the extreme values (smallest or largest) drawn from sufficiently large sub-samples, which in turn are drawn from a larger sample, tended to one of three possible unique asymptotic forms. For a given underlying distribution e.g. the exponential, the extremal distribution will be one of the three, in this case the Gumbel distribution (the others are Fréchet, to which the extremes of power laws are attracted, and the Weibull, also well known in failure rate modeling for example.) The probability density function of the GEV with location μ, scale σ, and shape ξ is:

$$P_{\mu,\sigma,\xi}(x) = \begin{cases} \exp\left(-\left(1+\xi\left(\frac{x-\mu}{\sigma}\right)\right)^{-1/\xi}\right) & \text{if } \xi \neq 0 \\ \exp\left(-e^{\left(\frac{x-\mu}{\sigma}\right)}\right) & \text{if } \xi = 0 \end{cases} \quad (1)$$

A detailed description, and physical examples of extreme value theory are presented in [21, 11, 31]. Next, we choose a common an simple parallel task (a write to a parallel file system) and argue that the i.i.d. assumption needed for GEV behavior are directly applicable as follows:

The storage nodes are independent. A storage node is here defined as a device that receives a portion of a file during a parallel write. While it is common to collect multiple devices into a storage array, our model treats an array as a single storage node that is independent from other arrays.

A write task takes place from a single node to many storage nodes. Of the many I/O scenarios enumerated in the article [25], this paper is concerned with the duration to complete *scenario 5: Checkpoint/restart with large I/O requests*. This is also known as a 'one-to-many' operation.

The dominant source of variation within the system arises from the storage nodes. The non-dominant sources of latency in the system including: network switches, network cards, interrupts, kernel buffers, PCI interfaces, OS schedulers, memory latency etc are all assumed to be comparatively small.

The client node is connected to each of the storage nodes by an identical network connection. The network connections connecting the client and storage nodes are identical in bandwidth and latency.

3. EXPERIMENT

A quantity of interest to many in HPC is the duration of time to complete a given task. Our chosen task is a write operation on a parallel file system with a duration of T_g. We assume that there is a baseline characteristic of the parallel task duration that is observable on a quiescent system without congestion T_s. Congestion is a important factor in network operations [12, 33, 9] that arises with a shared network or the storage nodes that are busy with other tasks. We encode the congestion penalty (which we call background traffic factor) as a constant of proportionality k_t. This gives: $T_g = k_t T_s$. A completely quiet system without congestion or background traffic is the state where $k_t = 1$. If background traffic is present, $k_t > 1$.

We extend our model with the assumptions: an observed file transfer to a *single* storage node will take S seconds where S is an observation of the storage node that behaves with a given probability distribution: $p(s)$. Hence the time taken T_s for the storage nodes to complete a *parallel* write in our model is the largest value of S from m storage nodes: $T_s = \max\{S_1, S_2...S_m\}$. By substitution, we arrive at:

$$T_g = k_t \max\{S_1, S_2...S_m\}. \quad (2)$$

i.e. a client will observe a write time onto a parallel file system that is limited by the last storage node to complete the task: $T_g = P_{\mu,\sigma,\xi}(x)$ from equation (1).

From Extreme Value Theory, provided m is sufficiently large and with our additional constant traffic constraint (k_t is constant across observations), we construct the following testable hypothesis: the times taken to transfer a file onto a large number of storage nodes will have a distribution approximated a random variable that has a extreme value distribution, given a fixed level of background traffic (congestion) and our previously stated assumptions of the system hold true.

An investigation to explore the distribution T_g was initially conducted at TACC on the Ranger system. Encouraging results were obtained. However, these results were identified as unreliable because the experimental run used `dd` with a block size of more than 2GB. For some configurations (apparently including Ranger), `dd` will stop writing after 2GB and return success. This initial data was discarded. An experimental run was subsequently completed on both Stampeed and Lonestar4 without success: these machines did not include the i.i.d. assumptions previously stated.

A second experiment was designed and conducted on the Amazon Web Services (AWS) public cloud. Cloud based computing has grown in popularity as a inexpensive tool for research, and performance evaluations are an area of active research [36, 6, 18, 34]. AWS allows dynamic construction of arbitrary configurations as well as isolated network environments - necessary to ensure constant k_t in our model. For a completely isolated network with a single client running a single job, $k_t = 1$.

Amazon Web Services provide basic specifications of the network and storage performance. They state a throughput of 128 MBps per volume [1], 62.5MBps per instance for write [2]. The dynamically constructed cluster was created within a 'placement group' [3]. This is a logical group of instances that enables applications to participate in a low-latency, 10Gbps network. Published values for the throughput of c3.large storage servers could not be obtained. The maximum theoretical bandwidth of a 10Gbps network is 1250 MBps. The mean value observed in our experiment is 45MBps. From these calculations it would appear that the instance throughput (possibly on the client) is the bottleneck in our system configuration.

Our experiments are performed on the Lustre[4] parallel file system version 1.8.9-wc1. While more recent Lustre software releases are available 1.8 is still a popular choice for production file systems [5] and has been the most common file system listed in previous variability papers. To avoid complications with caches, only synchronous write operations are considered in this study. The design of the Lustre file system version 1.8 requires a serialized meta-data request to open and close the file. We use a simple code (provided in the appendix) that measures the time for serialized meta-data requests separately to the parallel data transfer request. Our experiment defines a single write as a total file size of 512 MB written to 16 storage nodes. The default stripe size of 1MB was used. Choosing a files size of 512 MB ensures the file is small enough to fit in the client memory (total of 7.5GB) without needing costly swapping. 16 storage nodes is chosen as a sufficiently large population (m) and a total of 400 observations made to ensure sufficient fidelity of the underlying distribution and increase confidence of correct identification [17]

Specific compute instance (EC2) types and Elastic Block Store (EBS) were chosen as shown in Figure 1. The cluster was constructed behind a head node (not shown) in a private subnet within a placement group. The EC2 instances were

[1] http://aws.amazon.com/ebs/details/
[2] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-ec2-config.html
[3] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html
[4] Other names and brands may be claimed as the property of others.
[5] OpenSFS Lustre Usage Survey March 2015 http://wiki.opensfs.org/images/d/de/OpenSFS_Survey_Results_March_2015.pdf

shared tenancy. All instances in the experimental setup were CentOS 5.11 with Lustre 1.8.9-wc.

4. RESULTS

Figure 2 shows the duration of a parallel write is best approximated by equation (2). This results supports the hypotheses that the duration of a parallel write is controlled by the slowest node. GEV distributions are defined by three parameters: location, scale, and shape. The result of our work indicates that all three are valuable in capturing the variability characteristics of a system. HPC performance variability data first published in [19, 30] may now be better explained using the GEV model. [29] (and references therein) highlight the under appreciated importance, and poor level of understanding of variability, within cloud computing environments. Our results present a model that will provide for a deeper understanding of variability on both the cloud and HPC.

5. CONCLUSIONS

From extreme value theory, as the number of nodes increases we anticipate a universal behavior will emerge in systems of this type. We can confirm that with the conditions already stated, this is the case in our system (Figure 2). Our idealized experiment has wider implications as it maps onto a large class of systems, both physical and societal, where the essential element is waiting for a response in parallel from any nodes. In the computing field, for example, the Monte Carlo method is widely used and deployed at parallel scale and under certain configurations, the time to result would be expected to have a GEV distribution.

A complete, efficient, and accurate model of an HPC system is critical in optimizing utilization of this limited resource. Queues have already successful modelling a number of components of an HPC system including task scheduling [33, 32], network systems [20], and failure and recovery [4]. Our GEV model for parallel transfer grows the tools available to a model an entire, active, HPC cluster.

As high performance computing continues to develop and increase parallelism, new libraries become available (and necessary), to simplify interfacing with data objects [5]. For example, the `t3pio` library provides automatic configuration for MPI applications that use HDF5. With the GEV model, a library can be calibrated for ideal parallel (GEV) behavior and measure deviations from this behavior as values that are unlikely. The journey to exascale computing means vast increases node count and parallelism [1]. We expect GEV to be a powerful tool in understanding and exploiting variability on HPC systems in the future.

In summary, this paper explains the variability in parallel writes. The variability is explained by extreme value theory. Our analysis of data collected from a parallel write task performed in the public cloud found good agreement with well understood extreme statistics. Studies of parallel tasks should perhaps begin to consider examining repeated runs for evidence of extreme value distribution as a unique parallel performance signature.

6. ACKNOWLEDGMENTS

RH is grateful to Intel High Performance Data Division who supported this work. RH thanks various anonymous

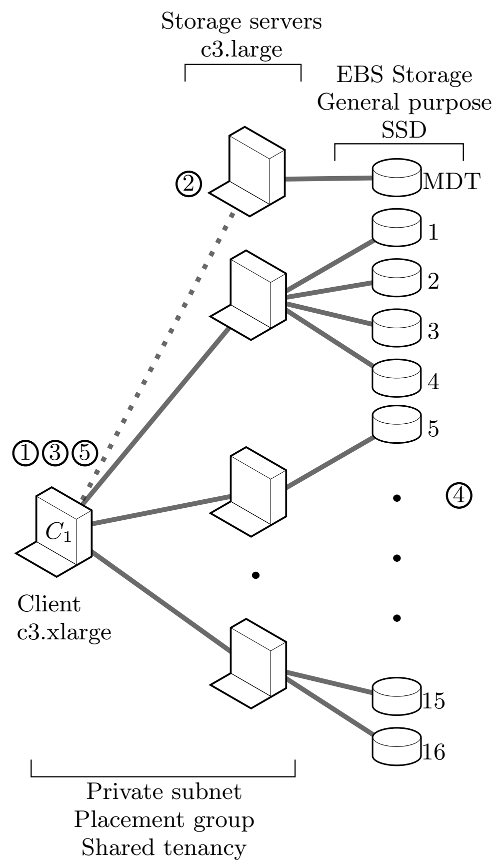

Figure 1: A typical high performance storage architecture with a single client node C_1. Storage targets (1-16) are attached in groups of four to storage servers. A read or write operation from C_1 occurs across all storage targets in parallel. A write operation includes the following high level steps: 1 C_1 executes a single task and accumulates results in memory until the task is complete. 2 C_1 requests a file handle from the metadata server. The metadata server persists data on storage (labelled 'MDT') and instructs the client to write to all the storage nodes during writing. From this point onwards the system storage targets behave with i.i.d. characteristics. 3 A timer begins on C_1. C_1 and the contents of the memory is written to all the storage nodes as a synchronous write. 4 The storage servers pass the data directly through to the EBS storage nodes (1-16). 5 The timer is stopped when C_1 is told that the write is complete. The value of the timer is T_g.

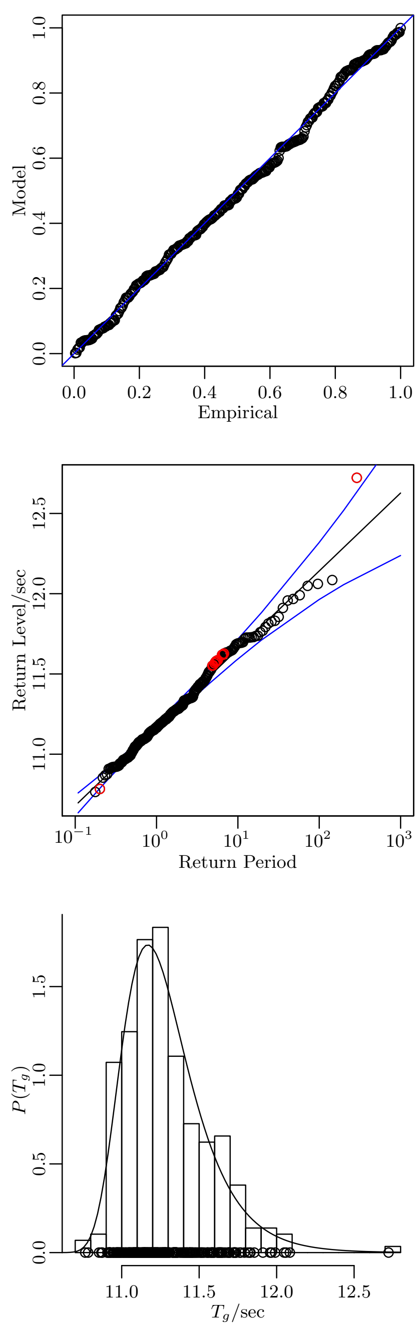

Figure 2: Parallel write times follow extreme statistics. 400 consecutive observations of T_g were taken. The top panel shows the cumulative value of the observation against the model value. The middle panel is the observed quantity plotted against the modeled quantity with the 95% confidence interval of the value of ξ shown as a blue line. Observations that fall outside the 95% confidence interval are colored in red. The bottom panel presents the observation histogram in 20 equal width bins with the fitted probability density over-plotted. The GEV fit has location $\mu = 11.1679 \pm 0.0140$, scale $\sigma = 0.2120 \pm 0.0101$, and shape $\xi = -0.00105 \pm 0.0415$. Values of μ, σ, ξ, standard errors, and outliers were calculated using the `ismev` library [16] within the R language environment [27].

reviewers who significantly improved this manuscript. SCC is supported by the UK EPSRC and STFC.

7. REFERENCES

[1] T. Agerwala. Exascale computing: The challenges and opportunities in the next decade. In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '10, pages 1–2, New York, NY, USA, 2010. ACM.

[2] L. Andersen and S. Asmussen. *Parallel Computing, Failure Recovery, and Extreme Values.* Stochastics series. T.N. Thiele Centre, University of Aarhus, 2007.

[3] S. Asmussen. Extreme value theory for queues via cycle maxima. *Extremes*, 1(2):137–168, 1998.

[4] S. Asmussen. *Importance Sampling for Failure Probabilities in Computing and Data Transmission.* Research report (Thiele Centre for Applied Mathematics in Natural Science). T.N. Thiele Centre, University of Aarhus, 2008.

[5] E. Barton. Lustre*-fast forward to exascale. *Lustre User Group Summit*, 2013.

[6] L. Bautista, A. Abran, and A. April. Design of a performance measurement framework for cloud computing. *Journal of Software Engineering and Applications*, 5:69, 2012.

[7] S. P. Bhavsar and J. D. Barrow. First ranked galaxies in groups and clusters. *Monthly Notices of the Royal Astronomical Society*, 213:857–869, Apr 1985.

[8] S. T. Bramwell. The distribution of spatially averaged critical properties. *Nat Phys*, 5:444–447, 2009.

[9] M. C. Calzarossa and E. Gelenbe. *Performance Tools and Applications to Networked Systems: Revised Tutorial Lectures (Lecture Notes in Computer Science).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2004.

[10] J. Choe and N. B. Shroff. New bounds and approximations using extreme value theory for the queue length distribution in high-speed networks., 1998.

[11] S. Coles. *An Introduction to Statistical Modeling of Extreme Values.* Springer Series in Statistics. Springer, 2001.

[12] J. Evans, W. Groop, and C. Hood. Exploring the relationship between parallel application run-time and network performance in clusters. In *Local Computer Networks, 2003. LCN '03. Proceedings. 28th Annual IEEE International Conference on*, pages 538–547, Oct 2003.

[13] R. A. Fisher and L. H. C. Tippett. Limiting forms of the frequency distribution of the largest or smallest member of a sample. *Mathematical Proceedings of the Cambridge Philosophical Society*, 24:180–190, 3 1928.

[14] P. W. Glynn and W. Whitt. Heavy-traffic extreme-value limits for queues. *Operations Research Letters*, 18(3):107 – 111, 1995.

[15] A. GÃşmez-Corral. On extreme values of orbit lengths in m/g/1 queues with constant retrial rate. *OR-Spektrum*, 23(3):395–409, 2001.

[16] J. E. Heffernan, A. G. Stephenson, and G. E. *ismev: An Introduction to Statistical Modeling of Extreme Values*, 2012.

[17] R. Henwood. ian investigation into recurrence in greenwich photoheliographic results 1874-1976. Master's thesis, Center for Fusion, Space and Astrophysics, University of Warwick, 2008.

[18] A. Iosup, S. Ostermann, M. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *Parallel and Distributed Systems, IEEE Transactions on*, 22(6):931–945, June 2011.

[19] W. Kramer and C. Ryan. Performance variability of highly parallel architectures. In P. Sloot, D. Abramson, A. Bogdanov, Y. Gorbachev, J. Dongarra, and A. Zomaya, editors, *Computational Science âĂŤ ICCS 2003*, volume 2659 of *Lecture Notes in Computer Science*, pages 560–569. Springer Berlin Heidelberg, 2003.

[20] I. Lahyani, N. Khabou, and M. Jmaiel. QoS Monitoring and Analysis Approach for Publish/Subscribe Systems Deployed on MANET. In *Proceedings of the 2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, PDP '12, pages 120–124, Washington, DC, USA, 2012.

[21] R. Leadbetter, G. Lindgren, and H. Rootzén. *Extremes and related properties of random sequences and processes*. Springer series in statistics. Springer-Verlag, 1983.

[22] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf. Managing variability in the io performance of petascale storage systems. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE Computer Society, 2010.

[23] S. Minkevičius. On extreme values in open queueing networks. *Mathematical and Computer Modelling*, 50(7âĂŞ8):1058 – 1066, 2009.

[24] N. R. Moloney and J. Davidsen. Extreme value statistics in the solar wind: An application to correlated lÃl'vy processes. *Journal of Geophysical Research: Space Physics*, 115(A10):n/a–n/a, 2010. A10114.

[25] H. Newman. HPCS Mission Partner File I/O Scenarios, Revision 3., 2008.

[26] K. K. Pusukuri, R. Gupta, and L. N. Bhuyan. Thread tranquilizer: Dynamically reducing performance variation. *ACM Trans. Archit. Code Optim.*, 8(4):46:1–46:21, Jan. 2012.

[27] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012.

[28] B. Schroeder and G. A. Gibson. Disk failures in the real world: what does an mttf of 1,000,000 hours mean to you? In *Proceedings of the 5th USENIX conference on File and Storage Technologies*, FAST '07, Berkeley, CA, USA, 2007.

[29] M. Schwarzkopf, D. G. Murray, and S. Hand. The seven deadly sins of cloud computing research. In *Presented as part of the*. USENIX, Submitted.

[30] D. Skinner and W. Kramer. Understanding the causes of performance variability in hpc workloads. In *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*, pages 137–149, Oct 2005.

[31] D. Sornette. *Critical Phenomena in Natural Sciences: Chaos, Fractals, Selforganization, and Disorder : Concepts and Tools*. Springer Series in Synergetics. Springer, 2004.

[32] D. Terekhov, T. T. Tran, D. G. Down, and J. C. Beck. Integrating queueing theory and scheduling for dynamic scheduling problems. *Submitted for publication*, 2013.

[33] A. Thomasian. Data allocation and scheduling in disks and disk arrays. In M. Calzarossa and E. Gelenbe, editors, *Performance Tools and Applications to Networked Systems*, volume 2965 of *Lecture Notes in Computer Science*, pages 357–384. Springer Berlin Heidelberg, 2004.

[34] G. Wang and T. S. E. Ng. The impact of virtualization on network performance of amazon ec2 data center. In *Proceedings of the 29th Conference on Information Communications*, INFOCOM'10, pages 1163–1171, Piscataway, NJ, USA, 2010. IEEE Press.

[35] N. Wright, S. Smallen, C. Olschanowsky, J. Hayes, and A. Snavely. Measuring and understanding variation in benchmark performance. In *DoD High Performance Computing Modernization Program Users Group Conference (HPCMP-UGC), 2009*, pages 438–443, June 2009.

[36] J. Yao, A. Ng, S. Chen, D. Liu, C. Friedrich, and S. Nepal. A performance evaluation of public cloud using tpc-c. In A. Ghose, H. Zhu, Q. Yu, A. Delis, Q. Sheng, O. Perrin, J. Wang, and Y. Wang, editors, *Service-Oriented Computing - ICSOC 2012 Workshops*, volume 7759 of *Lecture Notes in Computer Science*, pages 3–13. Springer Berlin Heidelberg, 2013.

APPENDIX

[Supporting Information: data write code]

```c
/* Code to observe GEV variability. Typically, 100 runs should
 * generate sufficient data for GEV to be confidently observed.
 * Example values:
 *      $FILESIZE = 512 // in MiB
 *      $STRIPECOUNT = 16
 *      $TARGETDIR = /mnt/lustre // on a Lustre file system.
 *
 * Ensure striping is set
 *
 *      lfs setstripe -c ${STRIPECOUNT} -s 1M $TARGETDIR"
 *
 * Run with:
 *
 *      ./timed_write $FILESIZE $TOPDIR/$runNum.dat
 *
 * Values of 'write' result, are expected to have GEV distribution.
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>

int main(int argc, char *argv[])
{
        int fd;
        int rc;
        unsigned char *bytes;
        long long write_size_mb;
        struct timeval start, end;
        long long calloc_elapsed_l, free_elapsed_l;
        long long open_elapsed_l, write_elapsed_l, close_elapsed_l;
        write_size_mb = atoll(argv[1]) * 1024 * 1024;

        printf("allocating local memory for write of %llu bytes\n", write_size_mb);

        /* gettimeofday has some known limitations:
         * http://stackoverflow.com/questions/88/
         * However, it should be sufficiently reliable for this experiment. */
        gettimeofday(&end, NULL);
        bytes  = calloc(write_size_mb, sizeof(unsigned char));
        if (bytes == NULL) {
                printf("can't allocate %s MiB: '%s'\n", argv[1], strerror(errno));
                return 1;
        }

        gettimeofday(&start, NULL);
        calloc_elapsed_l = (start.tv_sec * 1000000 + start.tv_usec) -
                (end.tv_sec*1000000 + end.tv_usec);

        gettimeofday(&start, NULL);
        fd = open(argv[2], O_SYNC|O_WRONLY|O_CREAT, 0644);
        if (fd < 0) {
                printf("can't open file: '%s' error: '%s'\n", argv[2], strerror(errno));
                return 1;
        }
```

```
        gettimeofday(&end, NULL);
        open_elapsed_l = (end.tv_sec * 1000000 + end.tv_usec) -
                (start.tv_sec*1000000 + start.tv_usec);

        rc = write(fd, bytes, write_size_mb * sizeof(unsigned char));
        if (rc < 0) {
                printf("can't write to this file: '%s' error: '%s'\n", argv[2], strerror(errno));
                return 1;
        }
        gettimeofday(&start, NULL);
        write_elapsed_l = (start.tv_sec * 1000000 + start.tv_usec) -
                (end.tv_sec*1000000 + end.tv_usec);

        close(fd);
        if (fd < 0) {
                printf("can't close file: '%s' error: %s\n", argv[2], strerror(errno));
                return 1;
        }
        gettimeofday(&end, NULL);
        close_elapsed_l = (end.tv_sec * 1000000 + end.tv_usec) -
                (start.tv_sec*1000000 + start.tv_usec);

        free(bytes);
        gettimeofday(&start, NULL);
        free_elapsed_l = (start.tv_sec * 1000000 + start.tv_usec) -
                (end.tv_sec*1000000 + end.tv_usec);

    printf("write complete: calloc %lf open %lf write %lf close %lf free %lf total %lf\n",
                calloc_elapsed_l/1000000.0,
                open_elapsed_l/1000000.0,
                write_elapsed_l/1000000.0,
                close_elapsed_l/1000000.0,
                free_elapsed_l/1000000.0,
                (calloc_elapsed_l
                        + open_elapsed_l
                        + write_elapsed_l
                        + close_elapsed_l
                        + free_elapsed_l)/1000000.0);

        return (0);
}
```