PROCEEDINGS OF SCIENCE

Speeding up Science Through Parametric Optimization on HPC Clusters

Jonas Weßner,[a,*] **Rüdiger Berlich,**[b] **Kilian Schwarz**[c] **and Matthias F. M. Lutz**[a]

[a] *GSI Helmholtz Center for Heavy Ion Research, Planckstrasse 1, Darmstadt, Germany*

[b] *Gemfony Scientific UG, Hermann-von-Helmholtz-Platz 6, Eggenstein-Leopoldshafen, Germany*

[c] *Hochschule Darmstadt University of Applied Sciences, Haardtring 100, Darmstadt, Germany*

E-mail: j.wessner@gsi.de, r.berlich@gemfony.eu, kilian.schwarz@h-da.de, m.lutz@gsi.de

Science is constantly encountering parametric optimization problems whose computer-aided solutions require enormous resources. At the same time, there is a trend towards the development of increasingly powerful computer clusters. Geneva is currently one of the best available frameworks for distributed optimization of large-scale problems with highly nonlinear quality surfaces. It is an excellent tool to be used in wide-area networks such as Grids and Clouds. However, it was not user-friendly for scheduling on high-performance computing clusters and supercomputers. Another issue was that it only provided a framework for parallelizing workloads at the population level of optimization algorithms, but did not support distributed parallelization of the cost function itself. For this reason, a new software component for network communication – called MPI-Consumer – has been developed and published as open-source software. As a complement to our previous paper, which explained the MPI Consumer's system design, this article provides an extensive performance evaluation. Comparing the MPI Consumer with two earlier in Geneva developed network technologies shows that the MPI Consumer is an overall improvement in terms of performance. Furthermore, we analyze the impact of system components of the MPI Consumer by testing different configurations. It is observed that asynchronous client requests speed up the time to solution by up to 20%. Furthermore, the multithreading design proves to be very scalable, allowing for a significant speed-up even if on extremely high loads. Tests on GSI's Green IT Cube HPC cluster show that our measurements realistically reflect the expected behavior in a production environment.

International Symposium on Grids and Clouds (ISGC) 2023,
19 - 31 March 2023
Academia Sinica Taipei, Taiwan

*Speaker

PoS(ISGC&HEPiX2023)004

1. Introduction

The Geneva [1][2] library is currently one of the best available frameworks for distributed optimization of large-scale problems with highly nonlinear quality surfaces. Recently, we have developed the *MPI Consumer*, a new network component for the Geneva optimization library, which increases its user experience for high-performance computing and also adds the new feature of fine-grained parallelization. We have discussed the design decisions of the MPI Consumer and also shown first performance test results on a 128-core machine in the previous publication [3]. Since this paper builds on top of our last paper, it is recommended to read the last paper before reading this one. To provide a comprehensive performance evaluation of the MPI Consumer, further test results are discussed in this paper. To begin with, we make a performance comparison of the MPI Consumer with the two other network components in the Geneva library (the Boost.Asio and the Boost.Beast Consumer) in Section 2. After that in Section 3, we analyze the impact of the async client requests and the server-side multithreading, which have been explained in our previous paper in detail, by measuring the performance of the MPI Consumer with different configurations. To ensure that our measurements from the previous sections are also accurate for clusters, we repeat a subset of the tests on a large HPC cluster and compare the results in Section 4. Finally, we conclude with a summary of our findings in Section 5 and future research topics in Section 6.

2. MPI Consumer Performance Compared to Other Consumers

In our previous work [3], we have shown a plot of a series of tests measuring the performance of the MPI Consumer with different numbers of clients and different durations of the user-defined cost function. Now, we have repeated these tests with the same configuration [3, p. 13] for the two alternative network components of the Geneva library, which have been developed earlier. In this section, we compare the test results of the MPI Consumer with those of the other two consumers.

The first networked consumer developed for the Geneva library is the Boost.Asio Consumer and has been available since 2018. This consumer uses the Boost.Asio C++ network library [4] to distribute the candidate solutions among the set of computers. Thereby, the server constantly listens for new TCP connection requests from clients. Once connected, clients send their processed work item to the server and the server responds with a new raw work item [5, 5-6]. The connection is then closed and the clients compute the cost function for the received work item. Once the cost function has been evaluated, the clients establish a new TCP connection to the server and repeat the above steps. During large-scale production runs for hadron physics research on GSI's Green IT Cube HPC cluster, limited scalability has been observed when using a small number of CPU cores or short cost function. This behavior is also visible in our performance test results. As in our previous paper, we have plotted the speedup, which is calculated as the quotient of serial execution time (one client) and the parallel execution time. Ideally, one would like to achieve linear scalability where the speedup is equal to the number of clients, i.e. multiplying the number of clients by x speeds up the time to solution by a factor x.

As can be seen in Figure 1, the Boost.Asio Consumer comes close to the desired behavior for up to 200 clients. As more clients are used, the performance does not further increase but drops again. This phenomenon becomes more severe for shorter cost functions. The limited scalability is most likely the cause of the need for opening new TCP connections for each work item: as the number of clients increases and the cost function evaluation time decreases, the frequency of incoming connection requests at the server increases. Eventually, the server is overloaded with the TCP handshakes and not able to answer the requests adequately.

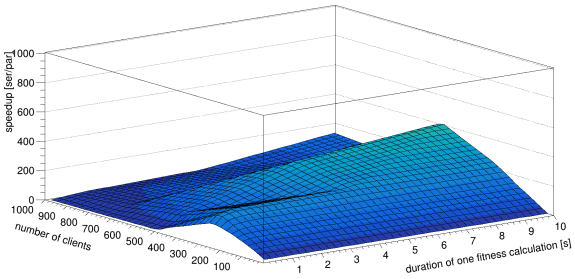

Figure 1: Results of performance tests evaluating the Boost.Asio Consumer's scalability on a 128-core computer. The speedup is calculated as the quotient of serial and parallel time to solution.

To provide an improvement over the Boost.Asio Consumer in terms of scalability, the Geneva team developed the Boost.Beast Consumer shortly after. This network component works very similarly but uses the WebSocket protocol [6] on top of TCP to communicate between server and clients. It does not open new connections between server and clients for delivering every work item. Instead, the clients open one persistent connection to the server at the time of startup.

The test results using the Boost.Beast Consumer are shown in Figure 2. As can be clearly seen, the scalability of the system is improved compared to using the Boost.Asio Consumer. With a cost function evaluation time of 5 seconds or more, the system reaches near-perfect scalability and an efficiency of $\approx 91\%$[1]. Note that perfect scalability i.e. efficiency of 100% is not feasible, since sending and receiving of work items and executing the optimization algorithm are non-parallelizable computations on the server that limit the scalability of the system according to Amdahls's law [7]. For the same reason, the system is less scalable when using cost functions that take less time to compute, since this decreases the length of the parallelizable computations and thereby increases the impact of non-parallelizable sections. We think that the scalability of the Boost.Beast Consumer is very reasonable. In our production runs for hadron physics at GSI, we regularly work with cost functions that take about 5 seconds to evaluate. Of course, if we work with shorter cost functions, the (serial) execution time will also decrease, so fewer clients will be needed to finish the optimization process in an acceptable time. For instance, for physics research at GSI we regularly require 1000 iterations and 10000 candidate solutions per iteration, where evaluating candidates takes about 5 seconds. If

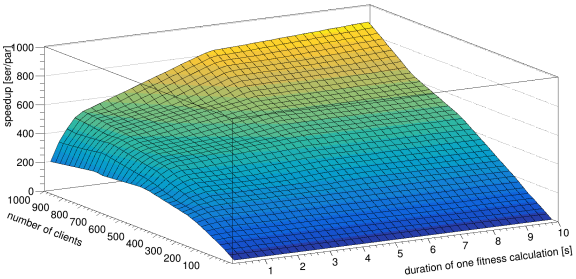

Figure 2: Results of performance tests evaluating the Boost.Beast Consumer's scalability on a 128-Core computer. The speedup is calculated as the quotient of serial and parallel execution time.

[1]Efficiency is calculated as the quotient of ideal execution time (serial execution time divided by the number of clients) and actual execution time.

done serially, this would take about $1000 \cdot 10000 \cdot 5 \cdot \frac{1}{60 \cdot 60 \cdot 24} \approx 579$ days. For parallel execution with 1000 clients and 90% efficiency, the execution time can be reduced to ≈ 15 hours (divide execution time by 900). Similarly, if the cost function evaluation time is 0.5 seconds, the user can achieve the same parallel execution time with 100 clients, since this configuration achieves a speedup of > 90, as can be seen in Figure 2.

The latest network component, the MPI Consumer, which we have developed in 2022, improves the usability of Geneva for high-performance computing and allows for fine-grained user-defined parallelization of the cost function. We have already shown the results of the performance tests for this consumer in our last paper [3]. Figure 3 illustrates the relative performance improvement provided by the MPI Consumer over the Boost.Beast Consumer. A z-value of x% (or $-x$%) means that the time to solution decreases (or increases) by x%. Note that for better visibility, we have used symmetric-logarithmic scaling for the z-axis.

One can see that the performance of the MPI Consumer is similar to that of the Boost.Beast Consumer when working with cost functions that take 5 seconds or longer to compute. For shorter cost functions, the MPI Consumer is a significant improvement over the already very efficiently working Boost.Beast Consumer, reducing the time-to-result by up to 3000%. Only for very short cost functions and more than 500 clients is the Boost.Beast consumer slightly faster. But as shown earlier, even with this configuration, the Boost.Beast Consumer only achieves poor performance with about 20% efficiency. For this reason, this part of the graph is irrelevant, as a user should not use more than 300 clients with either consumer when the cost function evaluation time is shorter than 1 second.

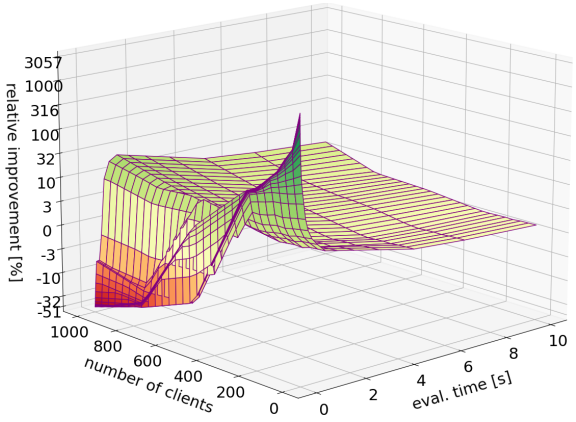

Figure 3: Relative improvement of time to solution when using the MPI Consumer instead of the Boost.Beast Consumer

3. Evaluating the Effect of the MPI Consumer's Design

In our previous publication [3], we explained the system design of the MPI Consumer in detail. The two features (1) asynchronous client requests and (2) multithreading have been implemented specifically to achieve great scalability. Having seen the test results in Section 2, we want to examine how the two features contribute to the high scalability. The asynchronous client requests can be disabled using the command line parameter −−mpi_worker_asyncRequests. To evaluate the impact of the asynchronous request feature, we have repeated the test series from Section 2 for the MPI Consumer with this feature deactivated.

In Figure 4, we have visualized the improvement from using asynchronous requests compared to using synchronous requests by plotting the relative difference of the time to solution of optimizations. Analogous to Figure 3, a z-value of x% (or $-x$%) indicates that the time to solution is decreased (or increased) by x% when using asynchronous requests instead of synchronous requests. As can be seen, the asynchronous request feature has an overall positive effect, reducing the time required for optimization by up to 20%. We also observe that the effect increases with shorter cost functions. The reason for this is that with a shorter cost function, the time required for network communication takes up a larger portion of the total runtime. Since asynchronous requests are an optimization to reduce the time needed for communication, it is more impactful when the time spent on computing cost functions is shorter. Similarly, the impact of the asynchronous requests is slightly greater if more clients are used. This is because additional clients place a heavier load on the server process, slowing its responses and increasing the time it takes a client to retrieve a new work item. The peak with one client and cost function evaluation time of 0.001 seconds is a single data point and has presumably been caused by concurrent multi-user access on the used computer. Overall, it can be concluded that a significant performance improvement was achieved by the asynchronous request feature. Looking back at Figure 3, we can see that the performance gain of up to 20% accounts for a large part of the improvements that the MPI consumer achieved compared to the Boost.Beast consumer. Moreover, we consider it positive that the feature has a stronger impact for shorter cost function evaluation times, since the system already runs with near-to-perfect when using longer cost functions, as shown in our previous paper in Section 5.2 [3].

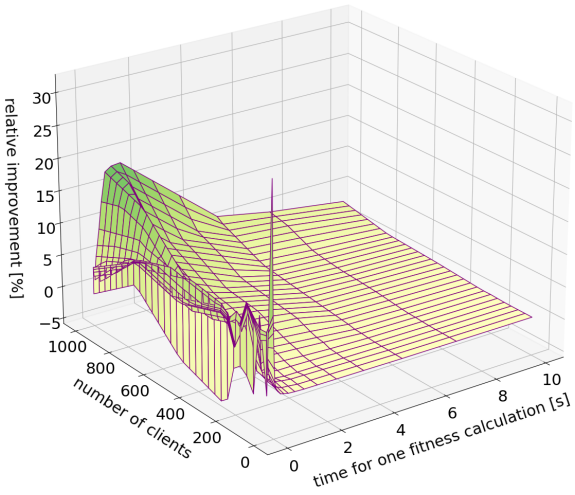

Figure 4: Relative improvement of the time to solution when using the MPI Consumer with its asynchronous request feature compared to using synchronous requests

The number of threads used by the MPI Consumer's server process can be configured with the command line parameter mpi_master_nIOThreads. To analyze the impact of the multithreading of the MPI Consumer server and evaluate the multithreading design described in our previous paper in Section 4.4 [3], we have run a series of tests with 400 clients and different numbers of threads as well as different lengths of cost functions on a 128-core machine. To illustrate the results, we have plotted the efficiency of the runs, calculated as the ratio between the ideal runtime and the actual runtime, in Figure 5. Again, note that an ideal runtime is not feasible due to communication overhead and non-parallelizable code fragments. One notices that more threads for the server process generally increase the system efficiency. However, for higher system efficiency, additional threads have less impact because there is less room for improvement.

With a cost function that takes 10 seconds to compute, the efficiency reaches ≈ 99% when the server process has 64 threads. Furthermore, even at maximum load on the server process, i.e. for a cost function with a length of 0.001 seconds, increasing the number of threads from 32 to 64 leads to a 53% efficiency gain. From this, we can conclude that the multithreading of the MPI consumer has not yet reached its limits. If the receiver thread was not able to accept and forward new connections fast enough, there would be a sudden limit where more threads would not lead to further efficiency gains. However, as we see gradual saturation, this is more indicative of a hardware-side limit to our test setup. Since the server process's threads and all 400 clients run on the same 128-core machine, the server threads are not continually scheduled on the CPU. For this reason, we assume that the MPI Consumer could perform even better than in our tests when used in production on a dedicated machine with one physical core per server thread.

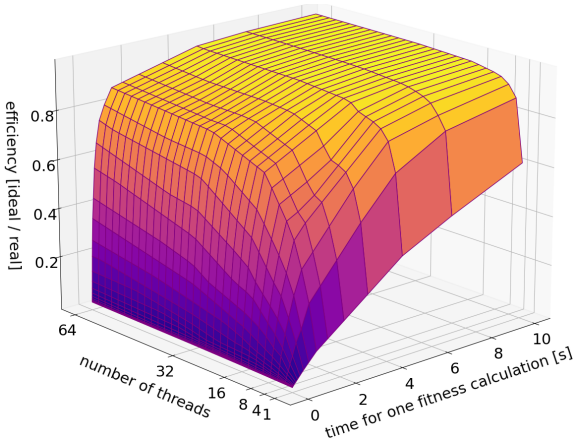

Figure 5: Efficiency of optimizations with MPI Consumer with different numbers of server threads and different lengths of the cost function evaluation time with 400 clients on a 128-core machine

4. MPI Consumer Performance on HPC Cluster

For cost reasons, we have executed the extensive test series shown in Section 2 and 3 on a single 128-core machine rather than on an HPC cluster. However, since the target platform for the MPI Consumer is clusters, we now repeat a subset of the tests on GSI's Green IT Cube supercomputer. The purpose of those tests is to see whether the test results shown earlier are reproducible on a cluster with reasonable error.

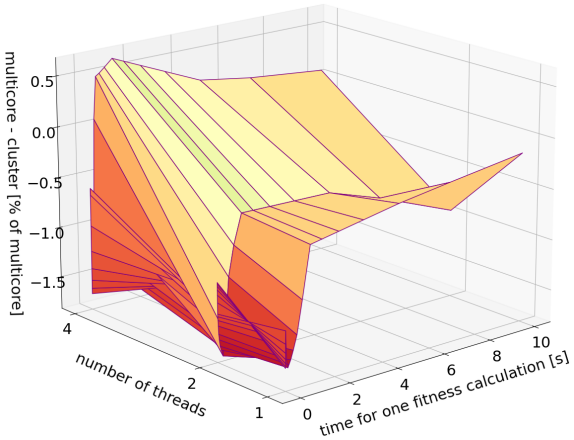

Figure 6: Difference of measurements on a 128-core computer and a cluster. Each data point is the mean value of 10 measurements.

We have chosen a test setup with a range of 1 to 4 server threads, 0.001 to 10 seconds for the cost function and 400 clients. For each configuration, we have run the tests on the multicore machine and on the cluster 10 times and calculated the mean value to ensure higher accuracy of the measurements. On both the multicore machine and the cluster, the mean absolute error for all data points is below 0.8 percent of the respective data point. Hence, we can assume that all calculated mean values are reasonably accurate. In Figure 6, we have visualized the difference in the mean values of measurements

on the multicore machine and on the cluster as a percentage relative to the measurements on the multicore machine. The plot shows that the measurements on the multicore computer are reproducible on the cluster with an error of below 1.5 percent. Thus, when running Geneva with the MPI Consumer in production on HPC clusters, we can expect a very similar behavior to that shown in the previous sections of this paper. We also notice that the time to solution on the cluster is slightly shorter when the cost function takes longer than 1 second to compute, and vice versa for shorter cost functions. A possible reason for the slightly shorter time to solution on the cluster could be that each process has its own CPU core in the cluster setup (as explained in Section 3). The marginally longer time to solution for short cost functions is most likely a cause of higher communication cost on the cluster due to slightly higher latency compared to a single machine. The reason is that with shorter cost functions and therefore less time spent on computation, communication between processes accounts for a larger portion of the runtime.

5. Conclusion

Our performance comparison of different network technologies in the Geneva optimization library has clearly shown the advantage of the Boost.Beast Consumer compared to the Boost.Asio consumer. The latest network component for HPC clusters and supercomputers, the MPI Consumer, performs even better than the Boost.Beast Consumer. By analyzing performance measurements using the MPI Consumer with different configurations, we have found that its asynchronous request feature can reduce the time to solution by up to about 20%. In addition, we have found that the multithreading design of the MPI Consumer is highly scalable, since even at maximum load, additional threads lead to a significant performance improvement. Finally, by repeating subsets of the tests on GSI's Green IT Cube supercomputer, we have observed that all previous test results are realistic for production runs on high-performance computing clusters.

6. Future Work

In a few months, the MPI consumer is expected to be brought into production on GSI's Green IT Cube supercomputer with 1000 clients or more. We also plan to implement new optimization algorithms and metrics for the Geneva optimization library.

References

[1] Gemfony Scientific. (2022) Geneva github repository. [Online]. Available: https://github.com/gemfony/geneva

[2] R. Berlich, S. Gabriel, and A. Garcıa. (2022) Parametric optimization with the geneva library collection - version: 1.6 (ivrea). [Online]. Available: http://www.gemfony.eu/fileadmin/documentation/geneva-manual.pdf

[3] J. Weßner, R. Berlich, K. Schwarz, and M. F. Lutz, "Parametric optimization on hpc clusters with geneva," 2023.

[4] C. M. Kohlhoff. (2020) Boost.asio homepage. [Online]. Available: https://www.boost.org/doc/libs/1_75_0/doc/html/boost_asio.html

[5] R. Berlich, S. Gabriel, and A. García, "Geneva 1.6: Improving the performance of highly concurrent workloads in parametric optimization," in *International Symposium on Grids and Clouds*, vol. 15, no. 20, 2015. [Online]. Available: https://pos.sissa.it/239/026/pdf

[6] I. Fette and A. Melnikov, "The websocket protocol," Tech. Rep., 2011.

[7] G. M. Amdahl, "Computer architecture and amdahl's law," *Computer*, vol. 46, no. 12, pp. 38–46, 2013. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6689270