Massive Contingency Analysis with High Performance Computing

Zhenyu Huang, *Senior Member, IEEE,* Yousu Chen, *Member, IEEE,* Jarek Nieplocha, *Member, IEEE*

***Abstract*—Contingency analysis is a key function in the Energy Management System (EMS) to assess the impact of various combinations of power system component failures based on state estimates. Contingency analysis is also extensively used in power market operation for feasibility test of market solutions. Faster analysis of more cases is required to safely and reliably operate today's power grids with less marginal and more intermittent renewable energy sources. Enabled by the latest development in the computer industry, high performance computing holds the promise of meet the need in the power industry. This paper investigates the potential of high performance computing for massive contingency analysis. The framework of "N-x" contingency analysis is established and computational load balancing schemes are studied and implemented with high performance computers. Case studies of massive 300,000-contingency-case analysis using the Western Electricity Coordinating Council power grid model are presented to illustrate the application of high performance computing and demonstrate the performance of the framework and computational load balancing schemes.**

***Index Terms*—Contingency Analysis, Energy Management System, Parallel Computing, Computational Load Balancing.**

I. Introduction

REAL time power grid operations heavily rely on computer simulation. A key function in the Energy Management System (EMS) is contingency analysis, which assesses the ability of the power grid to sustain various combinations of power grid component failures based on state estimates. The outputs of contingency analysis, together with other EMS functions, provide the basis for operation preventive and corrective actions. Contingency analysis is also extensively used in power market operation for feasibility test of market solutions.

Due to heavy computation involved, today's contingency analysis can be updated only every few minutes for only a select set of "N-1" contingency cases. A typical example is the EMS system at Bonneville Power Administration (BPA), one of the well-maintain system, runs 500 contingency cases in a time interval of five minutes. However, the trend of operating power grids closer to their capacity and integrating more and more intermittent renewable energy demands faster analysis of massive contingency cases to safely and reliably operate today's power grid.

One consequence of operating power grids closer to the edge is massive blackouts resulting in significant disruption of electricity supplies and economic losses [1][2]. Power grid blackouts often involve failure of multiple elements as revealed in recent examples. Preventing and mitigating blackouts requires "N-x" contingency analysis. The North American Electricity Reliability Corporation (NERC) moves to mandate contingency analysis from "N-1" to "N-x" in its grid operation standards [3]. All this calls for a massive number of contingency cases to be analyzed. As an example, the Western Electricity Coordinating Council (WECC) system has about 20,000 elements. Full "N-1" contingency analysis constitutes 20,000 cases, "N-2" is roughly 10^8 cases, and the number increase exponentially with "N-x".

It is obvious that the computational workload is beyond what a single personal computer can achieve within a reasonable time frame for real-time operation. Parallel computers or multi-core computers as emerging in the high performance computing (HPC) industry hold the promise of accelerating power grid contingency analysis. Contingency cases are relatively independent of one another, so contingency analysis is inherently a parallel process. Mathematically, there is a relatively straightforward parallelization path, but the issue with parallelization schemes remains due to the unevenness in computation time of individual cases.

Previous work in parallel computing for contingency analysis has been focused on "N-1" analysis with a small set of cases [4][5]. Scalability remains to be an issue when more processors are used and more cases are analyzed. This paper investigates the application of high performance computing for massive "N-x" contingency analysis. Parallelization schemes for computational load balancing of massive contingency analysis are investigated. The schemes include static load balancing scheme and task-counter based dynamic load balancing scheme. Test results are presented with actual WECC contingency cases. Superior linear scalability of parallel contingency analysis is demonstrated with 470 times speedup achieved for 150,000 and 300,000 WECC "N-1" and

This work is supported in part by the CASS-MT program funded by the Department of Defense and the Electricity Infrastructure Operations Initiative of the Pacific Northwest National Laboratory. The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under Contract DE-AC06-76RL01830.

Z. Huang, Y. Chen and J. Nieplocha are with Battelle – Pacific Northwest National Laboratory, Richland, WA 99352 USA (e-mails: zhenyu.huang@pnl.gov, yousu.chen@pnl.gov, and jarek.nieplocha@pnl.gov).

"N-2" contingency cases on a 512-processor parallel computer.

This paper starts with an overview of the trend in the HPC industry in Section II and the high performance computers used in our studies in Section III, followed by Section IV on the need of applying HPC to contingency analysis. Load balancing schemes, performance analysis, and actual case studies of massive contingency analysis are presented in Sections V, VI, and VII. Section VIII discusses relevant issues on contingency selection and decision support capabilities in the context of massive contingency analysis. Section IX concludes the paper with future work suggested.

II. Drive from High Performance Computing Technologies

Computer processor hardware has been significantly improved over the last decade from about 300MHz in 1997 to almost 4GHz today. However, looking back in the history (Figure 1), one can see that the single processor speed (i.e. clock frequency) is reaching a plateau and no longer follows Moore's Law [6], due to thermal limitations with the current CMOS process technologies. Therefore, the computational capacity of single core processors is no longer increasing much.

As technological limits on the clock speed of CMOS microprocessors are being approached, computer vendors are offering multiple-processor cores per socket while the performance of each microprocessor core remains relatively flat. Currently, microprocessors with eight cores per socket are available from Sun (multithreaded Sun Niagara [7]), four cores per socket from Intel, and two cores from AMD and IBM. Commodity computers with these multi-core processors are available in today's market. In addition, many-core processors are being developed. Experimental 80-core processors were announced by Intel and 128-core by IBM. Many-core processors are expected to enter the mainstream market in the near future.

Built with the multi-core/many-core processors, two major types of HPC architectures are available: shared-memory architecture and distributed-memory architecture. When applying HPC technologies, a key success factor is the match of computer architectures with problem characteristics. The shared memory architecture has the main memory block commonly accessible by all the processors in a random non-uniform manner. Many shared memory systems such as SGI Altix, Cray MTA-2, Cray XMT, or Sun Niagara provide a natural memory latency hiding capability and thus can efficiently execute applications with irregular memory references. Shared memory architecture is useful for efficient implementation of sparse matrix and irregular computations [8] such as power system state estimation. The distributed memory architecture consists of processors with local memory. High-speed data links are used between processors for communication. This architecture does not have the issue with main memory access, but inter-processor communication can be a bottleneck if an application requires frequent data exchange between processors. The distributed memory architecture well suits applications which can be divided into sub-tasks with minimum data communication requirements. Power system contingency analysis is one of such problems.

Early HPC applications to power system problems such as state estimation and contingency analysis have achieved promising results [9][10][11].

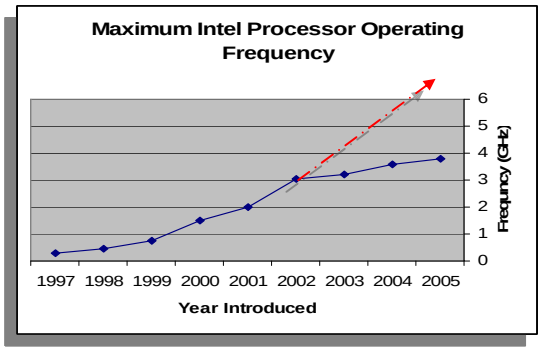


Figure 1 Increase of clock frequency of single-core Intel processors

III. High-Performance Computers and Parallel Programming Environment

Two high-performance computers – Colony2A and HP MPP2 – are used in this paper. They both are PC clusters with distributed memory.

Colony2A has 24 Itanium-2 computer nodes from Hewlett Packard. Each node has two 1.0 GHz processors (total 48 processors), 6 GB memory, and 36 GB disk space. Front-end login/compilation node is a 900 MHz Itanium-2 sinlge processor node with 1 GB memory. The network protocols include Myrinet-2000, Infiniband, Ethernet and GigE on all nodes. These provide high speed communication among the computing nodes.

The MPP2 machine consists of 980 Hewlett-Packard Longs Peak nodes with dual Intel 1.5 GHz Itanium-2 processors and HP's zx1 chipset. There are two types of nodes on the system: FatNodes with 10 GB of memory (5 GB per processor) and 430 GB of local disk space and ThinNodes with 10 GB of memory (5 GB per processor) and 10 GB of local disk space. Fast inter-processor communication is obtained using a single rail QSNetII/Elan-4 interconnect from Quadrics.

The program environment is Message Passing Interface (MPI). The MPI is a library specification for message-passing and a language-independent communications protocol used to program parallel computers. The message-passing model posits a set of processors that have only local memory but are able to communicate with other processors by sending and receiving messages, while each processor in the shared-memory model has access to all of a single, shared address space at the usual level of load and store operations [12].

IV. Need for High Performance Contingency Analysis

Contingency analysis is an essential part of power grid and market operations. Traditionally, contingency analysis is limited to be selected "N-1" cases within a balancing

authority's boundary. Power grid operators manage the system in a way that ensures *any single credible contingency will not propagate into a cascading blackout*, which approximately summarizes the "N-1" contingency standard established by the North American Electric Reliability Corporation (NERC) [13].

Though it has been a common industry practice, analysis based on limited "N-1" cases may not be adequate to assess the vulnerability of today's power grids due to new development in power grid and market operations.

On the power market side, one example is the introduction of Financial Transmission Rights (FTR) [14][15]. FTR provides market participants a means to hedge risks due to power transmission congestions. It is operated as an auction market. When clearing the FTR market, the feasibility of the FTR solution has to be evaluated by contingency analysis. There exist multiple FTR categories such as annual FTRs, seasonal FTRs and monthly FTRs. Each category requires contingency analysis of a full system-size model. Multiple categories couple the contingency problem and multiply the size of the model. The result is the number of cases is multiplied. With regular personal computers, it takes hours and even days to clear the FTR auction market.

As for power grid operation, recent cascading failures [2] reveal the need of "N-x" contingency analysis. The old assumption is that a cascading failure is caused by a single credible contingency. However, multiple unrelated events may occur in a system and result in cascading failures. Therefore, "N-2" and even higher order ("N-x") contingency events need to be analyzed.

Another new challenge in power grid operation is the separation of administrative boundaries – called Balancing Areas (or BAs) – which own, operate, and/or manage their own areas of the grid. When performing contingency analysis, each BA looks no further than its own boundaries. For areas within an interconnection where several BAs reside next to each other, seams issues may come into play. If the BAs all evaluate their system to be "OK" with the contingencies within their own systems, they will not prepare for the simultaneous occurrence of multiple contingencies. Individually, model results from each BA may show that each contingency does not cause a problem. However, if these contingencies occur simultaneously, there will likely be a very large system-wide impact, but the urgency to restore the system is not fully recognized with today's "N-1" contingency analysis. This indicates the need for "N-x" contingency analysis, i.e. analysis of simultaneous occurrence of multiple contingencies in multiple BAs.

"N-x" contingency analysis or even just more comprehensive "N-1" analysis is very challenging due to the combinatory number of contingencies and the extremely large amount of computational time. Our tests show that full "N-1" WECC contingency analysis has about 20,000 cases and takes about 15,000 seconds (~4 hours) to solve, and 150,000 "N-2" WECC cases take about 93,000 seconds (~26 hours). Obviously, high performance computing application is a must for meeting the need of massive power system contingency analysis. The performance of high-performance computing application for contingency analysis heavily relies on computational load balancing. A well-designed computational load balancing scheme considering the CPU speed, network bandwidth and data exchange latency is key to the success.

V. Computational Load Balancing Schemes for Massive Contingency Analysis

Contingency analysis is naturally a parallel process because multiple contingency cases can be easily divided onto multiple processors and communication between different processors is very minimal. Therefore, cluster-based parallel machines are well suited for contingency analysis. For the same reason, the challenge in parallel contingency analysis is not on the low-level algorithm parallelization but on the computational load balancing (task partitioning) to achieve the evenness of execution time for multiple processors.

The framework of parallel contingency analysis is shown in Figure 2. Each contingency case is essentially a power flow run. In our investigation, full Newton-Raphson power flow solution is implemented. Given a solved base case, each contingency updates its admittance matrix with an incremental change from the base case. One processor is designated as the master process (Proc 0 in Figure 2) to manage case allocation and load balancing, in addition to running contingency cases.

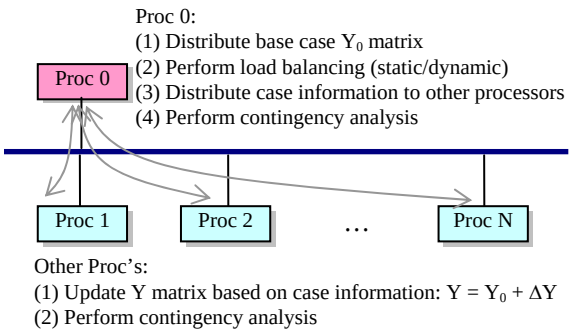


Figure 2 Framework of parallel contingency analysis

The straightforward load balancing of parallel contingency analysis is to pre-allocate equal number of cases to each processor, i.e. static load balancing. The master processor only needs to allocate the cases once at the beginning. Due to different convergence performance for different cases, the power flow run may require different number of iterations and thus take different time to finish. The extreme case would be non-converged cases which iterate until the maximum number of iterations is reached. The variations in execution time result in unevenness, and the overall computational efficiency is determined by the longest execution time of individual processors. Computational power is not fully utilized as many processors are idle while waiting for the last one to finish.

Another load balancing scheme is to allocate tasks to processors based on the availability of a processor, i.e. dynamic load balancing. In another word, the contingency cases are *dynamically* allocated to the individual processors so

that the cases are more evenly distributed in terms of execution time by significantly reducing processor idle time. The scheme is based on a shared task counter updated by atomic fetch-and-add operations. The master processor (Proc 0) does not distribute all the cases at the beginning. Instead, it maintains a task counter. Whenever a processor finishes its assigned case, the processor requests more tasks from the master processor and the task counter is updated. This process is illustrated in Figure 3. Different from the evenly-distributed number of cases on each processor with the static scheme, the number of cases on each processor with the dynamic scheme may not be equal, but the computation time on each processor is optimally equalized.

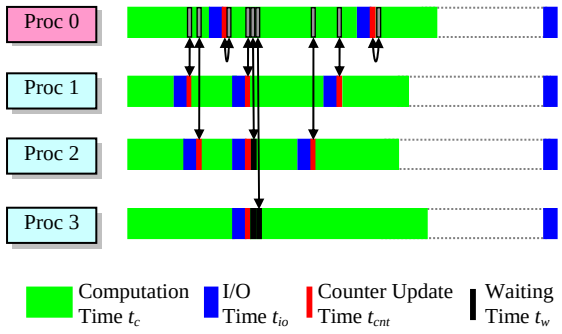


Figure 3 Task-counter-based dynamic computational load balancing scheme

Both computational load balancing schemes are tested with 512 "N-1" contingency cases of the 14,000-bus WECC system on the Colony2A machine. The results are shown in Figure 4. For the static scheme, though computational efficiency continuously increases when more processors are used, the performance is not scalable and exhibits the tendency of saturation. The performance of dynamic load balancing, in comparison with its static counterpart, shows much better linear scalability. The dynamic scheme achieves eight times more speedup with 32 processors as shown in Figure 4, and the difference is expected to be greater with more processors.

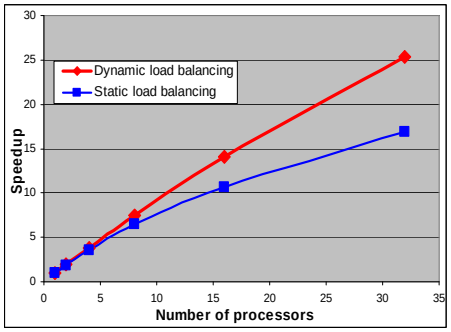


Figure 4 Performance comparison of static and dynamic computation load balancing schemes with 512 WECC contingency cases

Figure 5 further compares the processor execution time for the case with 32 processors. With dynamic load balancing, the execution time for all the processors is within a small variation of the average 23.4 seconds, while static load balancing has variations as large as 20 seconds or 86%. The dynamic load balancing scheme successfully improves speedups. It is also worth pointing out that the contingency analysis process of 512 WECC cases with full Newton-Raphson power flow solutions can be finished within about 25 seconds. It is a significant improvement compared to several minutes in current industry practice.

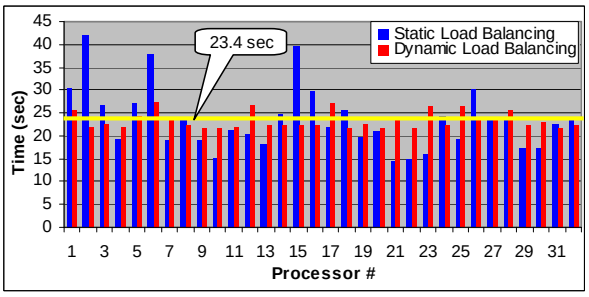


Figure 5 Evenness of execution time with different computational load balancing schemes

VI. Computational Performance Analysis of the Dynamic Load Balancing Scheme

The dynamic computational load balancing scheme balances execution time among processors better than the static scheme. But the cost is the overhead of managing the task counter. As shown in Figure 3, the execution time of each case consists of four parts: t_c – the computation time spent on solving one contingency case, t_{io} – the I/O time used to write the results to disks, t_{cnt} – the time to update the task counter, and t_w – the time to wait for the master processor to respond with a new case assignment when counter congestion occurs.

Running all the cases on only one processor would take a total time as estimated in (1):

$$t_{total} = \sum_{i=1}^{N_C} \left(t_c^{(i)} + t_{io}^{(i)} \right) = N_C \left(\bar{t}_c + \bar{t}_{io} \right) \qquad (1)$$

where N_C is the total number of cases, and $\bar{t}_c$ and $\bar{t}_{io}$ are the average computation time and I/O time, respectively. On one processor, there is no counter management needed, so no t_{cnt} and t_w should be included in (1).

Running the cases on multiple processors with dynamic load balancing scheme would evenly distribute the total time in (1), but involves counter management. If the total number of processor is N_P, the worst-case scenario with counter congestion is that all the N_P counter updates arrive at the same time at the master processor. Then the first processor has not waiting time, the second waits for time t_w, and the last one has the longest waiting time $(N_P - 1)t_w$. The average waiting time of a processor can be estimated as:

$$t_{w,N_P} \approx \frac{\frac{N_C}{N_P} \sum_{i=1}^{N_P} (i-1) t_w}{N_P} = \frac{N_C}{N_P} \frac{(N_P - 1) t_w}{2} \qquad (2)$$

Therefore, the total wall clock time required to run all the contingency cases can be estimated as (3):

$$t_{total,N_P} \approx \frac{\sum_{i=1}^{N_C}\left(t_c^{(i)} + t_{io}^{(i)}\right)}{N_P} + \frac{N_C}{N_P}t_{cnt} + \frac{N_C}{N_P}\frac{(N_P-1)t_w}{2} \quad (3)$$
$$= \frac{N_C}{N_P}\left(\bar{t}_c + \bar{t}_{io} + t_{cnt} + \frac{(N_P-1)t_w}{2}\right)$$

The speedup performance of dynamic load balancing scheme can be expressed as the following conservative estimate:

$$S_{N_P} = \frac{t_{total}}{t_{total,N_P}}$$
$$= \frac{N_C\left(\bar{t}_c + \bar{t}_{io}\right)}{\frac{N_C}{N_P}\left(\bar{t}_c + \bar{t}_{io} + t_{cnt} + \frac{(N_P-1)t_w}{2}\right)} \quad (4)$$
$$= N_P \frac{\left(\bar{t}_c + \bar{t}_{io}\right)}{\bar{t}_c + \bar{t}_{io} + t_{cnt} + \frac{(N_P-1)t_w}{2}}$$

Several observations can be drawn from (4):

1) It is clearly shown that the dynamic load balancing scheme is scalable with the number of cases as the speedup performance is irrelevant to the number of cases, N_C.
2) If the counter update is instantaneous and no counter congestion would occur, i.e. $t_{cnt} = 0$ and $t_w = 0$, then the ideal speedup performance would be N_P, equal to the number of processors.
3) For practical implementation, improving speedup performance would require to minimize the overhead t_{cnt} and t_w.
4) Counter update time t_{cnt} is mainly determined by the network bandwidth and speed. Minimizing t_{cnt} usually means to choose high-performance network connection between processors.
5) Waiting time t_w is due to counter congestion. Though more processors would improve the speedup, but they also increase the possibility of counter congestion as shown in (4).

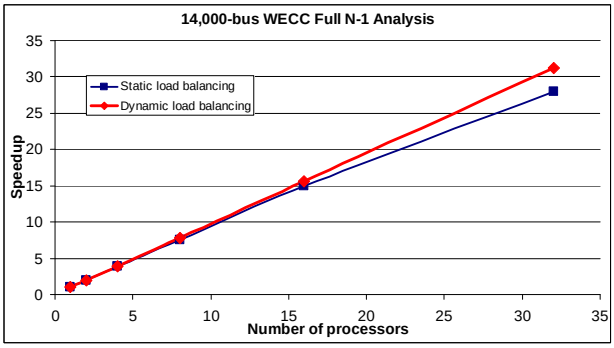
VII. Case Studies of Massive Contingency Analysis

The massive "N-x" contingency analysis framework with the dynamic computational load balancing scheme is implemented with both the Colony2A and MPP2 cluster machines. The 14,000-bus WECC power grid model is used to test the performance of the massive contingency analysis. Four scenarios of cases are selected for the studies:

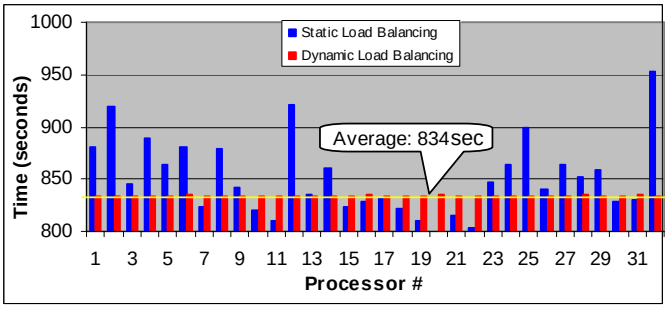
1) 20,094 full "N-1" cases, which consist of 2748 generator outage cases and 17346 line outage cases. Generator outages do not need to update the admittance matrix, but line outages do by adding an incremental change to the admittance matrix.
2) 150,000 "N-2" cases, which randomly choose 50,000 cases from each of the three combinations: double-generator outages, double-line outages, and generator-line outages.
3) 300,000 "N-2" cases, including 100,000 cases from the three combinations mentioned above.

The "N-2" scenarios have many more divergence cases as power flow would be more difficult to solve with double elements outaged.

Figure 6 shows the performance comparison of the Scenario 1 analysis on Colony2A with static and dynamic load balancing schemes. The dynamic load balancing scheme again exhibits superior speedup performance over the static one because the dynamic scheme perfectly balances execution time among all the processors as shown in Figure 6(b). Compared with Figure 4, the performance curve of the static scheme considerably approaches that of the dynamic scheme in Figure 6(a). This attributes to the increased number of cases on each processor: 16 (=512/32) in Figure 4 vs. 628 (=20,094/32). With more cases on each processor, the randomness effect with the static scheme tends to smooth out the unevenness in execution time. With more processors used and less number of cases on each processor, the two curves are expected to further depart, and the dynamic scheme will have much speedup performance than the static one.



(a)



(b)

Figure 6 Full WECC "N-1" contingency analysis on Colony2A (a) speedup performance and (b) execution time

The massive contingency analysis framework with the dynamic balancing scheme is further tested on the larger

MPP2 machine with all the scenarios including Scenario 1. The results are summarized in Table 1. 512 MPP2 processors are used, and excellent speedup performance is achieved: about 500 times with the "N-2" scenarios and slightly less with the "N-1" scenario. For all the scenarios, the counter time is very insignificant compared to the computation time. It indicates that the implementation of the dynamic balancing scheme using a counter adds very little overhead to the overall process and good scalability can be ensured.

TABLE 1 SUMMARY RESULTS OF THE MASSIVE CONTINGENCY ANALYSIS ON THE MPP2 MACHINE

512 Processors used	Wall Clock Time (seconds)	Total Computation Time (seconds)	Total I/O Time (seconds)	Total Counter Time* (seconds)	**Speedup**
Scenario 1 (20,094 N-1 cases)	31.0	14235.2	82.7	0.899	**462**
Scenario 2 (150,000 N-2 cases)	187.5	93115.5	489.1	5.550	**503**
Scenario 3 (300,000 N-2 cases)	447.9	226089.8	1087.1	9.984	**507**

* Includes waiting time.

As stated earlier, dynamic computational load balancing aims to balance execution time on each processor by dynamically distributing cases based on the availability of processors. This is cleared confirmed by Figure 8(a). It shows that the number of cases processed by individual processors varies from 20 to 50, but the computation stays flat across the processors. A noticeable larger time on Processor #51 is due to the last case on this processor being a diverged one and taking longer to solve.

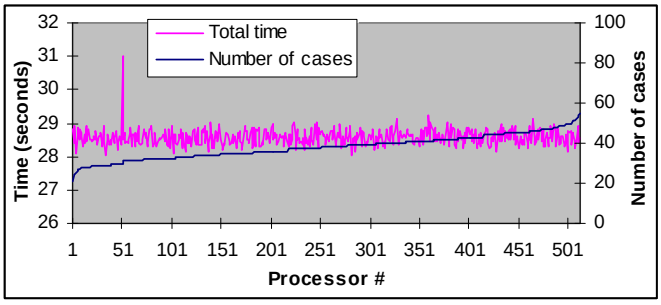


Figure 7 Total time vs the number of cases on each processor of Scenario 1

Figure 8(b) shows the increase in counter time with the increase in the number of cases on each processor for Scenario 1. This is understandable as more cases would need more counter update requests. For the MPP2 machine, the communication between processors is done with very high speed networking. One counter update takes about 10^{-5} seconds. The counter time is minimal compared to the computation time. But in other situations such as slow Ethernet-connected computers, the communication between processors is much slower and the counter time would be much larger. This would more likely cause counter congestion and should be taken into consideration when design the load balancing schemes.

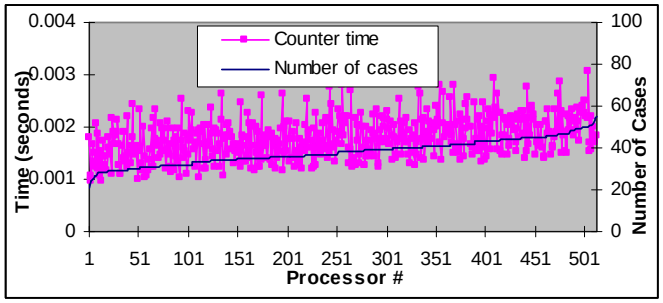


Figure 8 Counter time vs the number of cases on each processor of Scenario 1

VIII. DISCUSSION

The scalability of dynamic load balancing schemes with the number of processors is likely to be limited by counter congestion. More processors mean more counter update requests would be sent to the single master processor, which significantly increases the chances to have "traffic jam" on the path to the master processor. Ongoing work investigates multi-counter schemes and some other congestion-hiding schemes to minimize the impact of counter congestion.

Though HPC is expected to improve the computational efficiency of contingency analysis, smart contingency selection is still an important element of practical implementation. As mentioned in the Introduction section, the number of cases increases exponentially as the "x" in "N-x" increases. Even massive HPC application can not solve all the contingency cases. How to identify the credible "N-x" contingencies from a system-wide perspective can be a challenging task. Many of existing contingency ranking methods developed for "N-1" analysis [16] can be extended for "N-x" contingency selection.

High-performance contingency analysis calls for advanced operator decision support as much more information needs to be digested by operators within a short time periods of a few minutes. The technical challenge is how to navigate through the vast volume of data and help grid operators to manage the complexity of operations and decide among multiple choices of actions. The state-of-the-art industrial tools use tabular forms to present contingency analysis results. When there are only a few contingencies where the system is not "N-1" secure, the method of tabular display is adequate. But when massive "N-x" contingency cases are analyzed and the system is heavily stressed, the tabular method of display is rapidly overloaded by significantly more contingencies violations. It is then impossible for an operator to sift through the large amounts of violation data and understand the system situation within several seconds or minutes. However, it is in these situations that the operators most need the information while the tabular representation techniques are saturated. Thus the usefulness of massive contingency analysis is undermined and the HPC benefit is diminished. To resolve this issue, visualization and human factors can be good candidate approaches for enhancing decision support capabilities.

IX. Conclusions

A dynamic computational load balancing scheme is implemented using a shared task counter updated by atomic fetch-and-add operations to facilitate the work load management among processors for massive contingency analysis. The computational performance of the dynamic load balancing scheme is analyzed, and the results provide guidance in using high-performance computing machines for large number of relatively independent computational jobs such as power system contingency analysis. An "N-x" massive contingency analysis framework with the dynamic balancing scheme is tested on two different high performance cluster machines. The test results indicate excellent scalability of the dynamic load balancing scheme. On 512 processors, massive contingency analysis can achieve about 500 times speedup compared with a single processor, and full "N-1" WECC contingency analysis can be completed with half a minute.

Future work on computational load balancing will further focus on counter congestion management. Beyond load balancing, massive contingency analysis needs to study smart contingency screening and advanced decision support through techniques such as visualization.

X. Acknowledgement

The authors gratefully acknowledge productive discussions and dedicated support from Ning Zhou, Kevin Schneider, Daniel Chavarría, Robert Pratt, Carl Imhoff, and Jeffery Dagle, all with Pacific Northwest National Laboratory.

XI. References

[1] D. N. Kosterev, C. W. Taylor, and W. A. Mittelstadt, "Model Validation for the August 10, 1996 WSCC System Outage," *IEEE Trans. Power Syst.*, vol. 14, no. 3, pp. 967-979, August 1999.

[2] U.S.-Canada Power System Outage Task Force, "Final Report on the August 14, 2003 Blackout in the United State and Canada: Causes and Recommendations", April 2004. Available at https://reports.energy.gov/.

[3] NERC standards, Transmission System Standards – Normal and Emergency Conditions, available at www.nerc.com.

[4] Quirino Morante, Nadia Ranaldo, Alfredo Vaccaro, and Eugenio Zimeo, "Pervasive Grid for Large-Scale Power Systems Contingency Analysis," IEEE Transactions on Industrial Informatics, vol. 2, no. 3, August 2006

[5] Chen, R.H.; Jingde Gao; Malik, O.P.; Shi-Ying Wang; Nian-De Xiang; "Automatic contingency analysis and classification," The Fourth International Conference on Power System Control and Management, 16-18 April, 1996.

[6] Moore, Gordon E. "Cramming more components onto integrated circuits. Electronics", Vol. 38, No. 8, April 19, 1965.

[7] ---, "Niagara: A Torrent of Threads". Available at: http://www.aceshardware.com/reads.jsp?id=65000292

[8] T. Ungerer, B. Robi, and A. Jurij, "A survey of processors with explicit multithreading," *ACM Comput. Surv.*, vol. 35, pp. 29-63, 2003.

[9] Zhenyu Huang, and Jarek Nieplocha, "Transforming Power Grid Operations via High-Performance Computing," in: Proceedings of the IEEE Power and Energy Society General Meeting 2008, Pittsburgh, PA, USA, July 20-24, 2008.

[10] J. Nieplocha, A. Marquez, V. Tipparaju, D. Chavarría-Miranda, R. Guttromson, Zhenyu Huang, "Towards Efficient Power System State Estimators on Shared Memory Computers," in: Proceedings of the IEEE Power Engineering Society General Meeting 2006, Montreal, Canada, June 18-22, 2006.

[11] Zhenyu Huang, Ross Guttromson, Jarek Nieplocha and Rob Pratt, "Transforming Power Grid Operations via High-Performance Computing," Scientific Computing, April 2007.

[12] Williams Gropp, Ewing Lusk, and Anthony Skjellum, "Using MPI: Portable Parallel Programming with the Message-Passing Interface second edition", The MIT Press, 1999.

[13] ---, 'NERC Standards: Transmission System Standards – Normal and Emergency Conditions", North American Electricity Reliability Corporation. Available at: www.nerc.com.

[14] ---, "Financial Transmission Rights (FTR) and Auction Revenue Rights (ARR)", Midwest ISO, Mar 13, 2008.

[15] ---, "PJM eFTR Users Guide", PJM, 2007.

[16] Qiming Chen; McCalley, J.D.; Identifying high risk N-k contingencies for online security assessment, Power Systems, IEEE Transactions on, Volume 20, Issue 2, May 2005 Page(s):823 – 834.

XII. Biographies

Zhenyu Huang (M'01, SM'05) received his B. Eng. from Huazhong University of Science and Technology, Wuhan, China, and Ph.D. from Tsinghua University, Beijing, China, in 1994 and 1999 respectively. From 1998 to 2002, he conducted research at the University of Alberta, McGill University, and the University of Hong Kong. He is currently a senior research engineer at the Pacific Northwest National Laboratory, Richland, WA. His research interests include power system stability and control, high-performance computing applications, and power system signal processing.

Yousu Chen (M'07) received his B.E. in Electrical Engineering from Sichuan University, China, his M.S. in Electrical Engineering from Nankai University, China, and M.S. in Environmental Engineering from Washington State University. Currently he is a Research Engineer at the Pacific Northwest National Laboratory in Richland Washington. His main research areas are power system operations and high-performance computing applications. Mr. Chen is an IEEE member and the Vice-Chair of the Richland Chapter of the Power Engineering Society.

Jarek Nieplocha (M'??) is a laboratory fellow and the technical group leader of the Applied Computer Science Group in the Computational Sciences and Mathematics Division of the Fundamental Science Division at Pacific Northwest National Laboratory (PNNL). He is also the chief scientist for high-performance computing in the Computational Sciences and Mathematics Division. His area of research has been in optimizing performance of collective and one-sided communication operations on modern networks, runtime systems, parallel I/O, and scalable programming models. He received four best paper awards at conferences in high-performance computing: IPDPS '03, Supercomputing '98, IEEE High-Performance Distributed Computing HPDC-5, and IEEE Cluster '03 conference, and an R&D-100 award. He has authored and coauthored more than 80 peer reviewed papers. Dr. Nieplocha participated in the MPI Forum in defining the MP-2 standard. He is also a member of the editorial board of the International Journal of Computational Science and Engineering. He is a member of the IEEE Computer Society.