Real Time Contingency Analysis for Power Grids

Anshul Mittal[1], Jagabondhu Hazra[1], Nikhil Jain[2], Vivek Goyal[3], Deva P. Seetharam[1], and Yogish Sabharwal[1]

[1] IBM Research - India,
New Delhi, India
{mittal.anshul,jaghazra,dseetharam,ysabharwal}@in.ibm.com
[2] University of Illinois at Urbana-Champaign,
Illinois, USA
nikhil@illinois.edu
[3] IIT Delhi,
New Delhi, India
cs1070191@cse.iitd.ernet.in

Abstract. Modern power grids are continuously monitored by trained system operators equipped with sophisticated monitoring and control systems. Despite such precautionary measures, large blackouts, that affect more than a million consumers, occur quite frequently. To prevent such blackouts, it is important to perform high-order contingency analysis in real time. However, contingency analysis is computationally very expensive as many different combinations of power system component failures must be analyzed. Analyzing several million such possible combinations can take inordinately long time and it is not be possible for conventional systems to predict blackouts in time to take necessary corrective actions.

To address this issue, we present a scalable parallel implementation of a probabilistic contingency analysis scheme that processes only most severe and most probable contingencies. We evaluate our implementation by analyzing benchmark IEEE 300 bus and 118 bus test grids. We perform contingency analysis up to level eight (contingency chains of length eight) and can correctly predict blackouts in real time to a high degree of accuracy. To the best of our knowledge, this is the first implementation of real time contingency analysis beyond level two.

1 Introduction

Electric power systems are prone to various kinds of faults or disturbances. To withstand such disturbances, trained operators rely on computer simulations to continuously monitor the system and take corrective actions. However, large blackouts continue to occur across the globe[1]. For example, even in the US power grid having sophisticated controls, the frequency of blackout, which was about 7 per year until 1995, has grown to 36 per year in 2006[2]. Due to these blackouts, both the utilities and the consumers incur massive losses. According

E. Jeannot, R. Namyst, and J. Roman (Eds.): Euro-Par 2011, LNCS 6853, Part II, pp. 303–315, 2011.

to the US Department of Energy, 2003 US blackouts resulted in losses amounting to 6 billion USD[3]. Increasing frequency of severe blackouts indicate the need for tools that can reliably predict and prevent blackouts in real time.

One such tool is Contingency Analysis (CA), which assesses the ability of a grid to withstand cascading component failures/contingencies. The results of contingency analysis provide the basis for preventive and corrective operation actions against blackouts[1]. CA uses the current state reported by SCADA[1] or EMS[2] to identify possible series of component failures and check for collapse cases. The CA schemes are usually referred to as $(N-x)$ CA, where N is the total number of components (could be lines, generators and transformers) in the grid under consideration and x is the level/order. $(N-x)$ CA represents checking all possible permutations of x or less components (out of the total N) for a collapse. For example, a $(N-5)$ CA would evaluate all possible combinations of up to five components failing together in a cascade.

As the number of components (N) and number of levels (x) increase, the number of possible combinations that need to be evaluated increases exponentially ($\sum_{i=1}^{x} {}^{N}P_i$). Due to this computational complexity, contingency analysis has been traditionally limited to $(N-1)$ CA. However, post event analysis of major blackouts has shown that failing of a component leads to additional component outages in its vicinity. Moreover, the current trend of operating power grids closer to their capacity and integrating intermittent renewable energy sources has increased the probability of multiple component failures. Therefore performing higher order $(N-x)$ CA has become important. In fact, the North American Electricity Reliability Corporation (NERC) has recommended higher order CA as part of its Transmission Planning standards.

However, performing higher order CA for practical grids in real time (a few minutes) is not feasible using conventional techniques. Typically, a practical grid consists of a few thousands of components and even performing level 5 contingency analysis will involve a few billions of contingencies. Each contingency analysis takes about 50-100 ms on an ordinary computer. Hence, it is obvious that the computational workload is beyond what a single personal computer can achieve for real-time operation. This has lead researchers to turn to high performance computing platforms in order to accelerate power grid contingency analysis. The contingency analysis problem involves a large number of small independent computations. The challenge is not merely in parallelising it, but in doing so in real time. An important aspect here is to devise a load balancing scheme which scales to a large number of processors so that the full capabilities of a parallel system can be realised.

Our Contribution: In this paper, a parallel implementation of a probabilistic contingency analysis scheme, that processes only the most severe and most probable contingencies, has been developed. We have adopted search space reduction techniques to reduce the computational burden. We have also proposed a novel

[1] Supervisory Control and Data Acquisition.
[2] Energy Management System.

load balancing scheme which scales to thousands of processors. To the best of our knowledge, this is the first effort that goes beyond $(N-2)$ CA and scales well up to $8k$ processors.

The rest of this paper is organized as follows. Section 2 discusses previous work in this domain. Section 3 describes the risk based probabilistic approach and the algorithm used in this paper and the search space reduction techniques adopted. In Section 4, we introduce our novel load balancing scheme and provide a detailed comparison with the previous schemes. In Section 5, we present the results of our experiments. Finally in Section 6, we conclude the paper and propose some future work.

2 Previous Work

Contingency analysis in power systems was first proposed by Ejebe *et al*[4] in 1979. Since then several CA methods have been developed, each varying in methodology and complexity. However, they either employ approximate solution techniques, or use approximate models of the grid. Moreover, these methods are not suitable for higher order $(N-x,\ x>1)$ CA.

Recently, for higher order contingency analysis, Monte-Carlo simulation[5], Importance Sampling[6], and Risk Index (RI)[7] have been proposed. Monte-Carlo simulation and Importance Sampling techniques are not so efficient as they simulate the same set of contingencies repeatedly, delaying the convergence. RI approach, on the other hand, avoids repeated simulation and is much faster than Monte-Carlo simulation and Importance Sampling techniques.

Researchers have proposed several schemes to improve the computational speed of CA. Alves *et al*[8] proposed a parallel and distributed computing architecture for CA. For fast CA, Santos *et al*[9] developed a socket based client-server model where dynamic load balancing scheme is implemented for improved performance. Morante *et al*[10] developed a pervasive grid middleware which uses a broker system for reserving on-demand computational resources and for automatically splitting the contingency analysis task into sub-tasks and to allocate them to reserved resources based on a master-slave computing model. However, these and other methods focussed solely on $(N-1)$ analysis with a small set of cases. For massive higher order $(N-x; x \geq 2)$ contingency analysis, Huang *et al*[11] and Chen *et al*[12] proposed dynamic load balancing schemes to perform $(N-x)$ CA. However, their scheme naively selects either all or, a random subset of contingencies. Moreover, scalability remains to be an issue when more processors are used and more cases are analyzed. Most recently, Jin *et al*[13][14] proposed a CA approach using parallel betweenness centrality for contingency selection. This method identifies the most important lines based on base case power flow through lines and restricts higher order contingency analysis to those lines. This approach is overly conservative because it always considers the same set of contingencies as being critical. However, in case of multiple component outage, sets of critical contingencies dynamically change as power flow changes with each outage.

3 Risk Based Algorithm

In power grids, cascading failures happen in a chronological sequence. Therefore, it is convenient to model them with an event tree as shown in Fig. 1, where each node represents a state of the system and the branch between any two nodes represents a contingency. In the tree, the root node represents the pre-fault state of the system whereas a node with no forward branch represents an end node, i.e. a node which is either on the last level (level x), or represents a cascade leading to a blackout in the system.

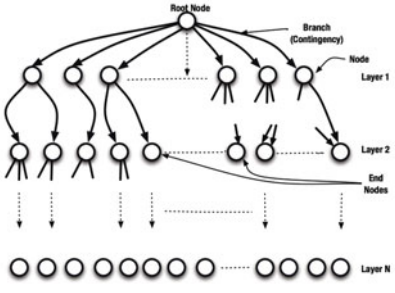


Fig. 1. Event tree

For any real grid, it is very difficult to explore the full event tree due to inordinately large number of possible paths. Therefore, CA schemes try to identify and traverse only those paths which may lead to a system collapse. In this paper, an intelligent search space reduction technique based on Risk Index (RI) is implemented. RI associated with each node is computed by multiplying the severity and probability of any contingency. Severity of any contingency is computed based on voltage instability, load loss, overload, available power margin, and frequency deviation as proposed in[7]. During event tree exploration, less relevant, low risk nodes are discarded at each node. This process is continued until the desired level is reached. RI captures both local and global information of the system, the probability part is computed based on local information whereas the severity part is computed based on global information.

We now propose a parallelization technique to make this algorithm suitable for execution in real-time (Algorithm 1). We compute N event trees (N being the number of lines), one for each line. Initially, the lines are (almost) equally divided amongst the processors. The processor responsible for an event tree begins the computation by simulating the tripping of the corresponding line; this corresponds to a real-life scenario wherein a natural event causes the line to trip, thereby triggering the breakdown process. A processor, say x, then determines the next set of elements (lines, generators, transformers) in the vicinity of this element that are most likely to get affected by the failure of this element. These are referred to as *exposed elements*. It creates a new child node in the event tree for each of the exposed elements. The processing of these child nodes is

Algorithm 1

```
Compute base case load flow for the current system state.
Select child processor, send first lines and base case matrix to it.
MPI_Irecv(RI response from children)
MPI_Irecv(A new contingency to evaluate)
while(1)
    Check end condition (whether all jobs completed or not)
        If yes, exit program.
    Test if RI response received from all children for a contingency, if yes,
        if(RI value is selected)
              MPI_Isend(go-ahead message to that child)
        else if(RI value is rejected)
              MPI_Isend(abort message to that child)
              goto beginning of while loop.
    Test if a new contingency received for evaluation, if yes,
        compute RI.
        If there is a system collapse
             report and put RI = ∞.
        MPI_Isend(RI value to parent)
        MPI_Irecv(go-ahead/abort message from parent)
        MPI_Irecv(A new contingency to evaluate)
    Test if (go-ahead/abort) message received from any parent, if yes,
        if(abort received or max level reached)
             goto beginning of while loop.
        else if(go-ahead received)
             find exposed elements; select children processors
             send new contingency and base case solution to the children.
             MPI_Irecv(RI response from children)
             goto beginning of while loop.
```

distributed to new processors. If there are m exposed elements, x selects m new processors and sends them an "RI-evaluation" request, handing over to them, the responsibility of performing the processing for each of these nodes. It then waits for the child nodes to compute and return the RI value for the nodes allocated to them. Once it receives the RI values from all the nodes, it selects the most risky elements based on a certain criteria (explained later) dependent on the RI values. It then sends a "go-ahead" message to the processors corresponding to nodes that are selected for further exploration and an "abort" message to the remaining processors. This completes the processing of the current node for x.

A processor may receive four types of messages, "RI-response", "RI-evaluation" request or, "abort" message or "go-ahead" message. Note that at any point of time, a processor may have requests corresponding to multiple event-tree nodes pending with it. These are queued by MPI; he processor receives one request at a time from MPI and handles it as shown in Algorithm 1. The messages are processed in the given order so as to give priority to those messages which free

up the memory. The end condition and process of selecting the children varies according to the load balancing scheme used and is discussed in detail in the next section.

The decision of whether or not to further explore a node in the event tree is based on the RI values. The objective here is to maximise the risk coverage, defined as follows:

$$RC_l = \frac{\sum_{i=1;i\in N}^{k} RI_i}{\sum_{i=1}^{N} RI_i} \times 100 \tag{1}$$

where RC_l is the percentage risk coverage up to layer l, N is the number of possible blackouts up to layer l, k is the number of blackouts identified by the proposed method and RI_i is the risk associated with the blackout sequence i. RI_i is calculated as follows:

$$RI_i = SI_i * p_{fault} \times \prod_{j\in tripped} p_{cj} \tag{2}$$

$$p_{cj} = p_j \times \prod_{k\in\ exposed\&not\ tripped;k\neq j} p_k \tag{3}$$

where SI_i and Pr_i are severity and probability of the blackout i respectively, p_{fault} is the probability of the initiating fault, p_{cj} is the conditional tripping probability of exposed equipment j in the blackout sequence i, and p_j is the tripping probability of equipment j.

The risk coverage, and hence, the effectiveness of the entire analysis, is heavily dependent on the choice of the number of event tree nodes to explore further. Some strategies to determine this parameter are (i) select all the child nodes to explore further, (ii) select a fixed percentage of the child nodes or (iii) select all the child nodes above a certain threshold of RI value. While it is desirable to explore all the child nodes, this leads to a significant computation load. It is also very difficult to obtain a single RI value to use as a thumb rule for the threshold as the RI values vary significantly depending on the specific contingency and the test case under consideration. While the option of selecting a fixed percentage looks promising, our experiments show that this strategy does not result in good risk coverage.

To address this issue, we devise a new strategy that combines a novel extension of the percentage selection strategy and the threshold based strategy. The idea is based on the observation that as the event tree grows exponentially with increasing levels (depth), we can afford to explore more nodes at lower levels (towards the top of the event tree) but fewer nodes at higher levels (towards the bottom of the tree). We therefore apply a linear function to determine the percentage of nodes to select for exploration; this linear function is set up so that it returns 100% at the top level and about 20% at the bottom-most level. Along with this, we also use a threshold value to rule out contingencies with very small RI values. This heuristic results in very good risk coverage (around 80%) and reduces the computation significantly (see Figure 2). 80% percentage risk coverage is fairly good for our application because remaining unidentified

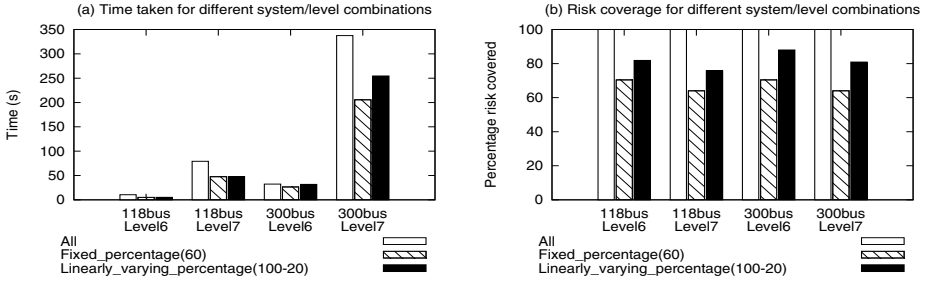


Fig. 2. Comparison of different next-element selection schemes

cascades are of very low probability and consist of longer chain of events. Hence, operator will get enough time to take preventive/corrective controls.

Another optimization is based on the observation that in most of the contingency chains that lead to a collapse, there was a significant jump (at least a 15%) in the RI value before the tripping of the last element in the contingency sequence. For instance, consider a contingency sequence L1-L2-L3-L4-L5 that leads to a collapse. Then, there would be an at least a 15% jump in the sequence RI1, RI2, RI3, RI4 corresponding to the risk indices for the contingencies L1, L1-L2, L1-L2-L3, L1-L2-L3-L4, respectively. This is not surprising since the tripping of the first few lines may cause some lines to become highly overloaded or a few generators may reach their limit leading to considerable worsening of the grid health. Therefore, at the penultimate level, we give a go-ahead to only those sequences that exhibit such a jump in the RI values. Since more than 90% of the contingencies are in the last level only, incorporating this optimization in the algorithm significantly reduces the computation load.

4 Load Balancing Schemes

The ratio of communication to computation per task should be minimised for a scheme to achieve good performance. In case of higher order CA, the number of tasks is huge (of the order of millions) and the task granularity (relative amount of computation per task) is very low (of the order of milliseconds). Therefore, any good load balancing scheme for this problem should involve minimum book keeping and communication amongst the processors, as even a small delay can reduce the communication to computation per task considerably.

Load balancing schemes can be broadly classified into two categories: (i) centralized and (ii) decentralized schemes. While centralized schemes offer better control over the load balance as all the information is available at a single node, decentralized schemes, in contrast, are less prone to congestion, particularly when the number of processors is very large.

Huang et al.[11] have implemented and compared several centralized load balancing schemes. However, these schemes are not directly applicable to the algorithm discussed in this paper due to the modular nature of our contingency

analysis scheme, where contingency selection and contingency evaluation are performed separately (Section 2). We have implemented a modification of these schemes for our algorithm. We first discuss them and then propose a new decentralized scheme. Our decentralized scheme does not require any book keeping and scales linearly with increasing number of processors. It is thus highly suited for contingency analysis when there are a large number of processors and the analysis is performed for higher levels.

4.1 Centralized Load Balancing Schemes

In the centralized schemes, a processor designated as master allocates tasks to others processors.Whenever a processor needs to spawn some tasks (corresponding to the child nodes in the event tree), it sends a request to the master indicating the number of tasks to be spawned. The master then, based on the scheme being used, assigns a set of processors and sends a list of these processors back to the requesting node. The master keeps track of the number of jobs spawned in the system and sends an end signal to all processors when all jobs have completed. We now discuss some of these schemes investigated in prior work.

Static allocation: In this scheme, the master ensures that every processor gets equal number of tasks by doing round robin based allocation. The amount of bookkeeping done at master is minimal. This scheme should work well for small system sizes as shown by Huang et al[11]. For larger system sizes, this scheme is expected to underperform as the advantages gained due to the minimal bookkeeping diminish.

Dynamic allocation: This scheme tries to equalize the computation load across the processors. In order to do this, the master maintains a list of active jobs on every processor and assigns request for new tasks to processors that are least loaded. This list is updated as tasks begin and finish on the processors. This involves a considerable amount of bookkeeping and therefore results in an increase in the computation time at the master. For large levels and large number of processors, this has the affect of increasing communication delays between the processors and the master as intermittently requests tend to get queued up at the master. While, this scheme should outperform the static allocation scheme on account of better load balancing, its performance deteriorates for large levels and system sizes.

Two master allocation: Chen et al[12] proposed a variant which they referred to as the multi counter based dynamic allocation. In this scheme, the inital task list is divided into multiple masters which allocate processors for new tasks using the dynamic approach. If the task list of any processor becomes empty, it steals them from the other master(s). The observed performance of this scheme has been found to be similar to previous scheme.

Though these schemes are expected to do well when the number of tasks is not too large and the levels are few, they suffer from congestion issues at the

master nodes and hence do not scale to a large number of tasks and larger levels of analysis.

4.2 Decentralized Load Balancing Scheme

Chen et al[12] suggest that if the allocation queries can be serviced instantaneously by the master then ideal speedup can be achieved in the centralized scheme with dynamic allocation. However, as number of tasks increase for larger levels, the congestion at the master causes the network queues to build up and the service time cannot be ignored anymore. In order to address this, we propose a decentralized load balancing scheme that aims at reducing the service time for new task requests while continuing to balance the computation load across all the processors. The master performs the bookkeeping primarily for two purposes; the first is to balance the load amongst the processors and the second is to declare completion of the processing. In our new scheme, we eliminate bookkeeping alltogether in order to enable decentralized control.

To handle the load balancing, we handle task allocation as follows. Whenever new tasks have to be spawned corresponding to the child nodes in the event tree, the processor handling the current (parent) node selects as many processors as the number of child nodes uniformly at random from the set of available processing nodes. It then sends the information regarding the task to be performed directly to the corresponding processors. As every processor makes local decisions regarding the set of processors to allocate the tasks to, the queries are serviced locally and hence instantaneously. There is no master involved in this scheme. If all the processors start with distinct initial seeds for the random number generation, it can be shown that when the number of tasks spawned is very large, the tasks are distributed over the processors uniformly with very small deviation. Hence for higher levels of contingency analysis, considering the granularity of the tasks and the number of tasks involved, the load imbalance is not expected to be high. This scheme is therefore expected to scale linearly with the number of processors as well as the levels of contingency analysis performed.

To detect completion of processing, the nodes perform a collective Allreduce operation at regular interval to determine the number of unfinished tasks. Completion is declared when all the processors report that there are no unfinished tasks remaining. The Allreduce is performed at an interval of $100t$ units where t represents the units taken to perform an Allreduce. This ensures that the overheads of completion detection are no more than 1% of the processing time.

5 Results

In this section, we evaluate our algorithm and compare it with previously studied algorithms.

Hardware setup. All implementations are on Blue Gene/P - IBM's massively parallel supercomputer; Each node of the Blue Gene/P system consists of four

850 MHz PowerPC 450 processor cores. Torus network handles the bulk of the communication data from an application and offers the highest bandwidth in the system. Each node supports 850 MBps bidirectional links to each of its nearest neighbors for a total of 5.1GB/s bidirectional bandwidth per node.

Test Cases. We evaluate the performance using the IEEE Standard test cases[15] comprising of 118 bus system containing 186 lines and 300 Bus System containing 411 lines.

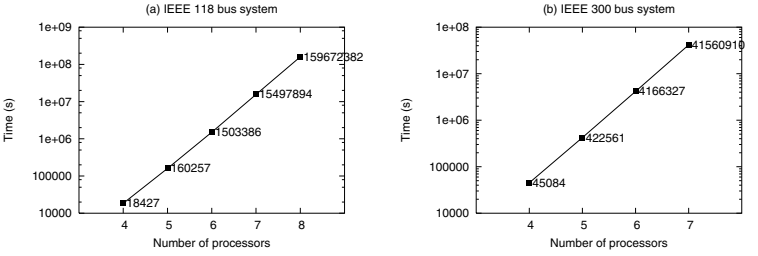
We present the number of contingency chains generated using the RI based selection technique for varying levels on IEEE 118 and IEEE 300 bus systems in Figure 3(1). These results conform to the expected exponential increase in the search space with increase in the number of levels explored. We observe a factor 10x increase in the number of contingency chains with every level (branching factor in event tree). The number of contingency chains runs in tens of millions for level 7 and hundreds of millions for level 8.

In Figure 3(2), results for comparative study of execution time of various load balancing schemes with varying levels are presented. The results show that the increase in execution time for our scheme is commensurate to increase in problem size. In contrast, for centralized schemes the execution time increases super-linearly for large problem sizes.

We study the scalability of our scheme with increase in system size in Figure 3(3). Our scheme outperforms both the centralized schemes for large system sizes. The increase in system size has a negative effect on scaling of both the centralized schemes. Our scheme, on the other hand, scales almost linearly. The gap in performance increases as the problem size increases (from level 4 to level 5). For level 5, the performance of our scheme is an order of magnitude better than the centralized schemes. These gains can be attributed to absence of wait queues at the master node. However, for small system sizes, static scheme outperforms both the dynamic scheme (due to large turnaround time) and our scheme (due to inefficient allocation of jobs). For level 6, the centralized schemes fail to complete successfully in certain cases; this is primarily attributed to the requests piling up on the master causing the processor to run out of memory.

In Figure 3(4), we present strong scaling results for our scheme to show its scalability to very large levels and very large system sizes. We report results for level 6 and 7 for which real time analysis has been made possible by our scheme even for medium system sizes like 512 and 1024 processors. Figure 3(4) shows that our scheme scales nearly linearly for large levels for large systems with up to $2k$ processors. It can also be seen that the scheme scales very well up to $8k$ processor systems. We obtain a factor 12 speedup for $8k$ processor system relative to 512 processor system. Along with the good speed up, it is worth noticing the fact that all these runs up to level 6 and 7 can be done in real time; in contrast, as of today no system goes beyond level 2 for online calculations.

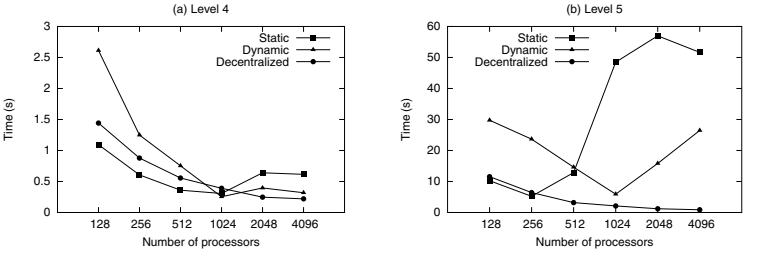
To test the real time nature and scalability of our scheme to highest level, we also ran the code for *level 8 on IEEE 118 bus system* and it took only *259 seconds on 8k processors*. The number of contingencies evaluated in this case is nearly 150 million. A serialized, or parallel centralized scheme based, contingency analysis



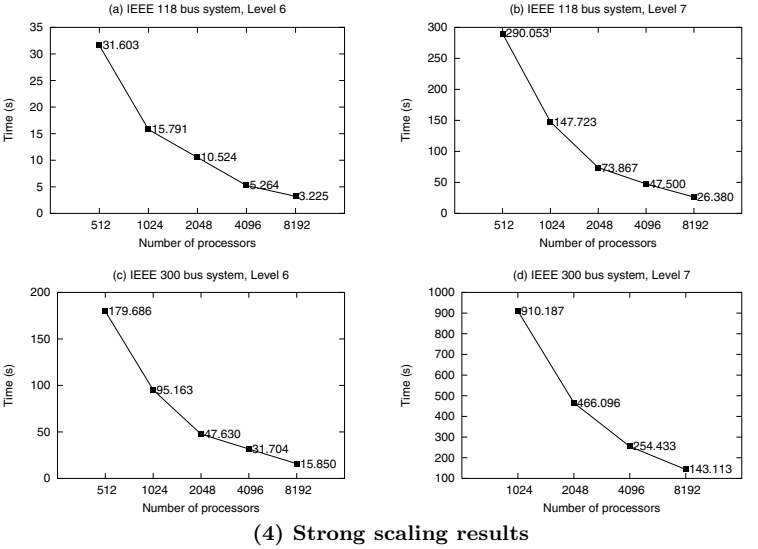
(1) Number of contingencies with change in levels

Level	Time (s)		
	Static	Dynamic	Decentralized
3	0.070	0.074	0.092
4	0.640	0.395	0.247
5	56.970	15.837	1.199

(2) Level wise comparison of load balancing schemes on 118 bus system



(3) Comparison of load balancing schemes on IEEE 118 bus system



(4) Strong scaling results

Fig. 3. Results with varying levels and system sizes

version will take several days to complete this analysis. The results indicate that our scheme scales well for large number of processors and outperforms the other load balancing schemes. The difference between the schemes becomes more prominent with increasing levels, due to the increasing load on the system and with increasing system sizes, due to the increasing difficulty in load balancing.

6 Conclusions and Future Work

We presented a parallel implementation of probabilistic real time contingency analysis scheme which could be used for blackout prediction in power grid. We evaluated up to 150 million contingencies and showed real time results up to level 8. We also presented a novel load balancing scheme achieving good scalability up to 8k processors. Future work includes incorporating transient stability analysis into this implementation and analyzing the performance on different machines.

References

1. Knight, U.G.: Power Systems in Emergencies: From Contingency Planning to Crisis Management. Wiley, New York (2000)
2. http://edition.cnn.com/2010/TECH/innovation/08/09/smart.grid/index.html
3. Bill Parks: Transforming the Grid to Revolutionize Electric Power in North America. In: U.S. Department of Energy, Edison Electric Institutes Fall 2003 Transmission, Distribution and Metering Conference (2003)
4. Ejebe, G.C., Wollenberg, B.F.: Automatic contingency selection. IEEE Trans. Power Apparatus and Systems PAS-98 (1), 92–104 (1979)
5. Xingbin, Y., Singh, C.: A practical approach for integrated power system vulnerability analysis with protection failures. IEEE Trans. Power Systems (2004)
6. Thorp, J.S., Phadke, A.G., Horowitz, S.H., Tamronglak, S.: Anatomy of power system disturbances: importance sampling. Int. J. Electr. Power Energy Syst. (1997)
7. Hazra, J., Sinha, A.K.: Identification of catastrophic failures in power system using pattern recognition and fuzzy estimation. IEEE Trans. Power System (2009)
8. Alves, A., Monticelli, A.: Parallel and distributed solutions for contingency analysis in EMS. In: Proc. of the Midwest Symposium on Circuits and Systems (1995)
9. Santos, J.R., Exposito, A.G., Ramos, J.L.M.: Distributed Contingency Analysis: Practical Issues. IEEE Trans. on Power Systems 14(4), 1349–1354 (1999)
10. Morante, Q., Ranaldo, N., Vaccaro, A., Zimeo, E.: Pervasive Grid for Large-Scale Power Systems Contingency Analysis. IEEE Transactions on Industrial Informatics 2(3) (August 2006)
11. Huang, Z., Chen, Y., Nieplocha, J.: Massive Contingency Analysis with High Performance Computing. In: Proc. PES General Meeting, Canada (July 2009)
12. Chen, Y., Huang, Z., Chavarra-Miranda, D.: Performance Evaluation of Counter-Based Dynamic Load Balancing Schemes for Massive Contingency Analysis with Different Computing Environments. In: Proc. of the 2010 IEEE PES General Meeting (2010)

13. Jin, S., Huang, Z., Chen, Y., Chavarra-Miranda, D., Feo, J.T., Wong, P.C.: A Novel Application of Parallel Betweenness Centrality to Power Grid Contingency Analysis. In: IPDPS 2010, pp. 1–7 (2010)
14. Gorton, I., Huang, Z., Chen, Y., Kalahar, B., Jin, S., Chavarra-Miranda, D., Baxter, D., Feo, J.T.: A High-Performance Hybrid Computing Approach to Massive Contingency Analysis in the Power Grid. In: Proc. of Fifth IEEE International Conference on e-Science (2009)
15. http://www.ee.washington.edu/research/pstca/