Joint Laboratory for Petascale Computation

INRIA NCSA

Redesiging fault tolerance for High performance Computing

Franck Cappello
INRIA and UIUC
Joint-Laboratory on Petascale Computing

I2PC Seminar

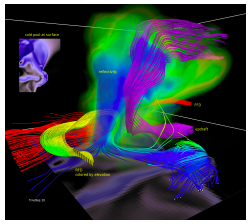
INRIA

NCSA

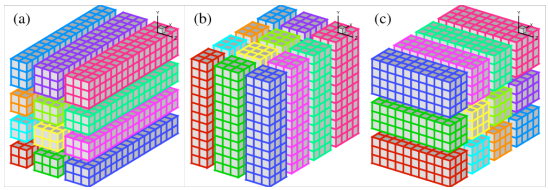
INRIA-Illinois Joint-Laboratory

- Google “joint-lab” → first link
  - Between INRIA and UIUC
  - In the context of Blue Waters
  - From 2009 and for 5 years
  - 2 co-executive directors: Marc Snir (UIUC) and Franck Cappello (INRIA)
  - 3 types of activities: workshops, visits, missions + *Collaborations: FR, EU, US, ASIA*
  - Research topics: Parallel programming, runtime, management software, numerical libraries, fault tolerance, files system and archive, etc.
  - Several achievements: ACM student award 2010, “Perfect Score Paper” at SC11, ~30 joint publications, G8 ECS, 45 visits in 2 years 1/2,

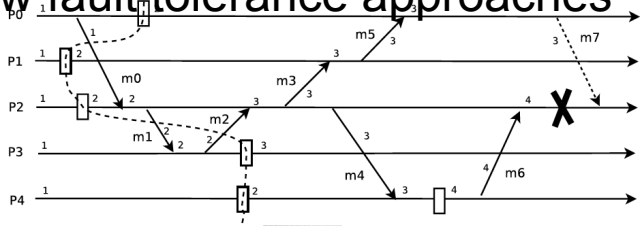
Tornado simulation



3D-FFT

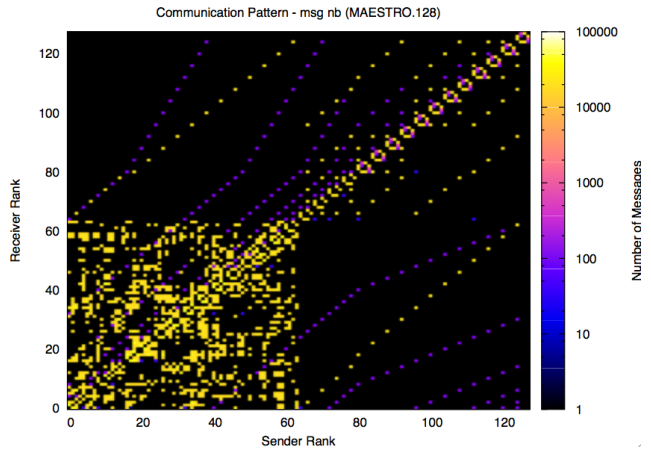


New fault tolerance approaches



Summary

- **Fault tolerance for message passing HPC applications**
- Exploring and exploiting application fundamental properties to improve fault tolerance protocols
- Analyzing system notifications to improve fault tolerance
- Conclusion

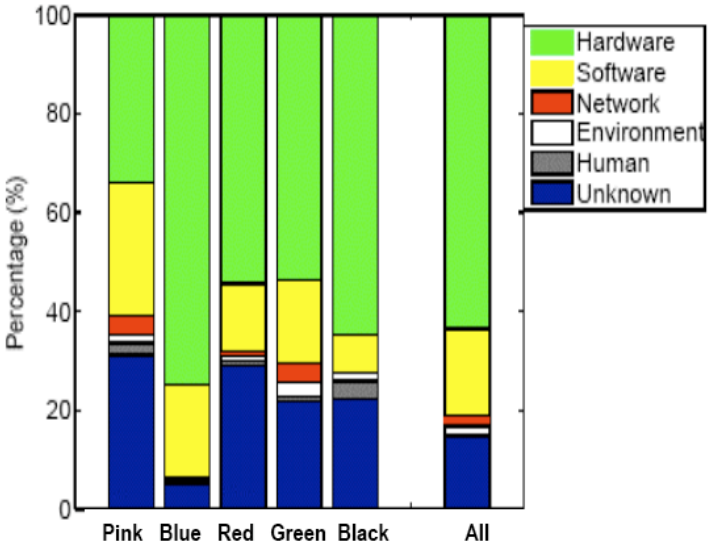


Current situation Fault Model and Approach for HPC

Types of faults:

- Power outage
- Hard errors (broken component: memory, network, core, disk, etc.)
- Detected soft errors (bit flip in memory, logic, bus)
- OS error (buffer overrun, deadlock, etc.)
- System Software error (service malfunctio
- Application bugs
- Administrator error (Human)
- User errors (Human)

G. Gibson



Relative frequency of root cause by system type.

Classes of faults:

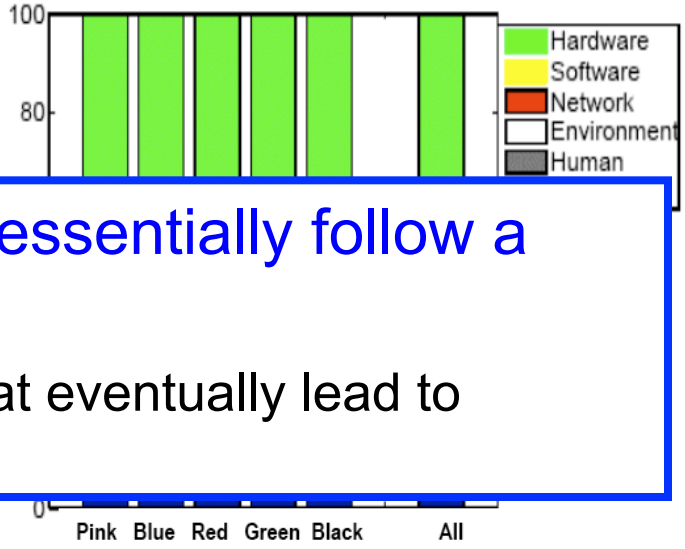
- Detected and corrected (by ECC, Replication, Re-execution)
- Detected and uncorrectable (leading to application crash: fail stop)
- Undetected (leading to data corruption, application hang, etc.)

Current situation
Fault Model and Approach for HPC

Types of faults:

- Power outage
- Hard errors (broken component: memory, network, core, disk, etc.)
- Detected soft errors (bit flip in memory
- Administrator error (Human)
- User errors (Human)

G. Gibson



Relative frequency of root cause by system type.

To tolerate these faults current systems essentially follow a masking approach:

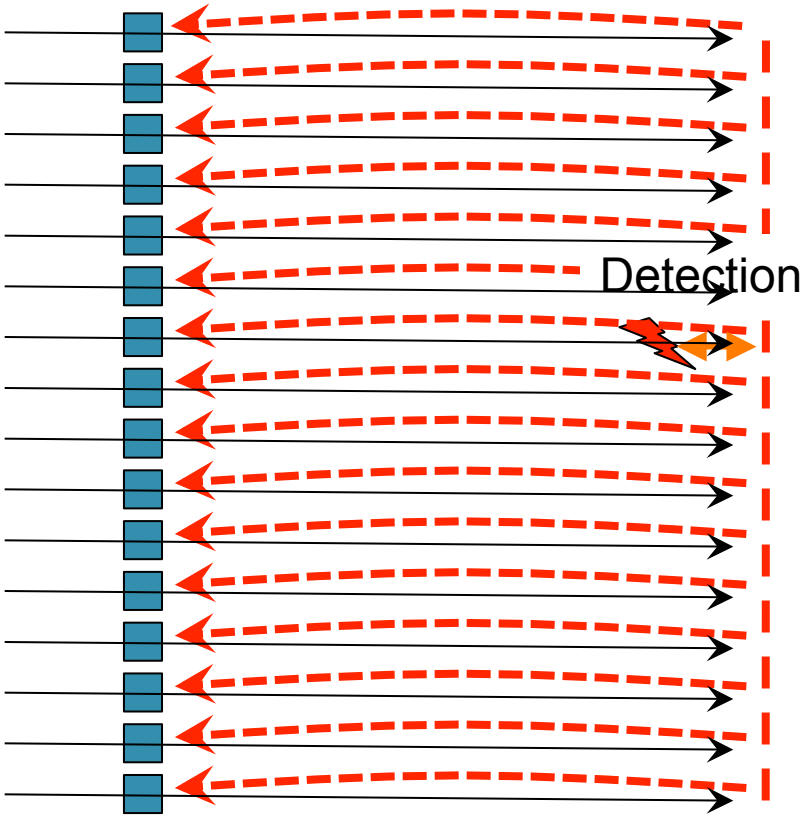
-Checkpoint-restart for process failure (errors that eventually lead to application crash)

Classes of faults:

- Detected and corrected (by ECC, Replication, Re-execution)
- Detected and uncorrectable (leading to application crash: fail stop)
- Undetected (leading to data corruption, application hang, etc.)

Rollback-recovery

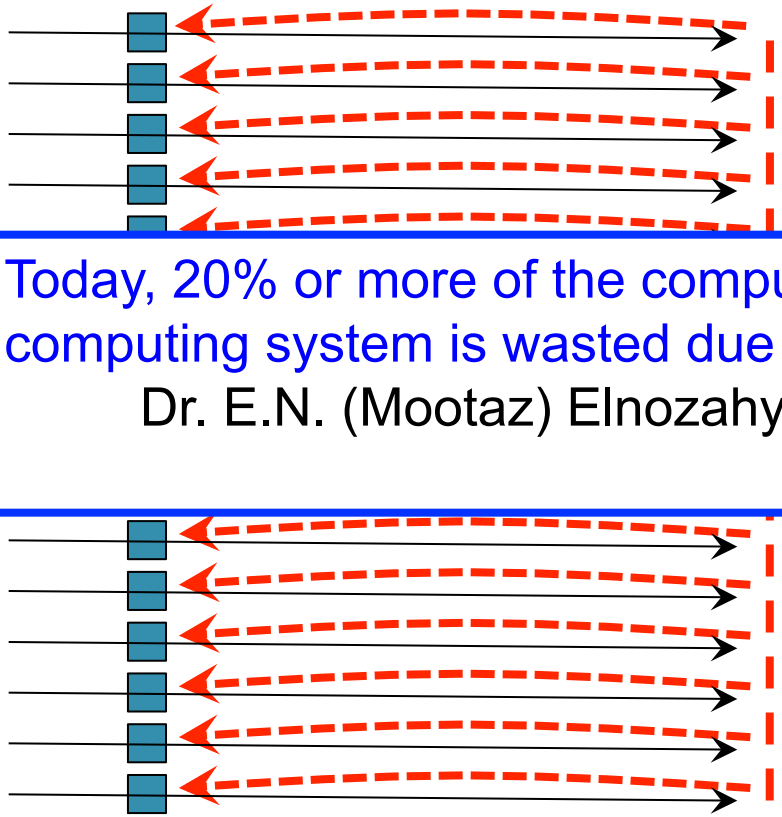
Full execution Checkpoint-Restart:



Systems	Perf.	Ckpt time	Source
RoadRunner	1PF	~20 min.	Panasas
LLNL BG/L	500 TF	>20 min.	LLNL
Argonne BG/P	500 TF	~30 min.	LLNL
Total SGI Altix	100 TF	~40 min.	estimation
IDRIS BG/P	100TF	30 min.	IDRIS

Rollback-recovery

Full execution Checkpoint-Restart:



Checkpoints Restart message

Systems	Perf.	Ckpt time	Source
IDRIS BG/P	100TF	30 min.	IDRIS

Today, 20% or more of the computing capacity in a large high-performance computing system is wasted due to failures and recoveries.

Dr. E.N. (Mootaz) Elnozahyet al. *System Resilience at Extreme Scale, DARPA*

Alternatives, and main issues of ckpt/rst

Joint Laboratory for Petascale Computation

INRIA NCSA

Alternatives approach:

- Applications may tolerate errors and failures to some extend
- Algorithmic techniques like ABFT can help making algorithms more robust.
- How to efficiently mix the different approaches (ex: ckpt and ABFT) efficiently is not know

The last resort will be checkpoint/restart

- Checkpoint coordination is not really a problem:

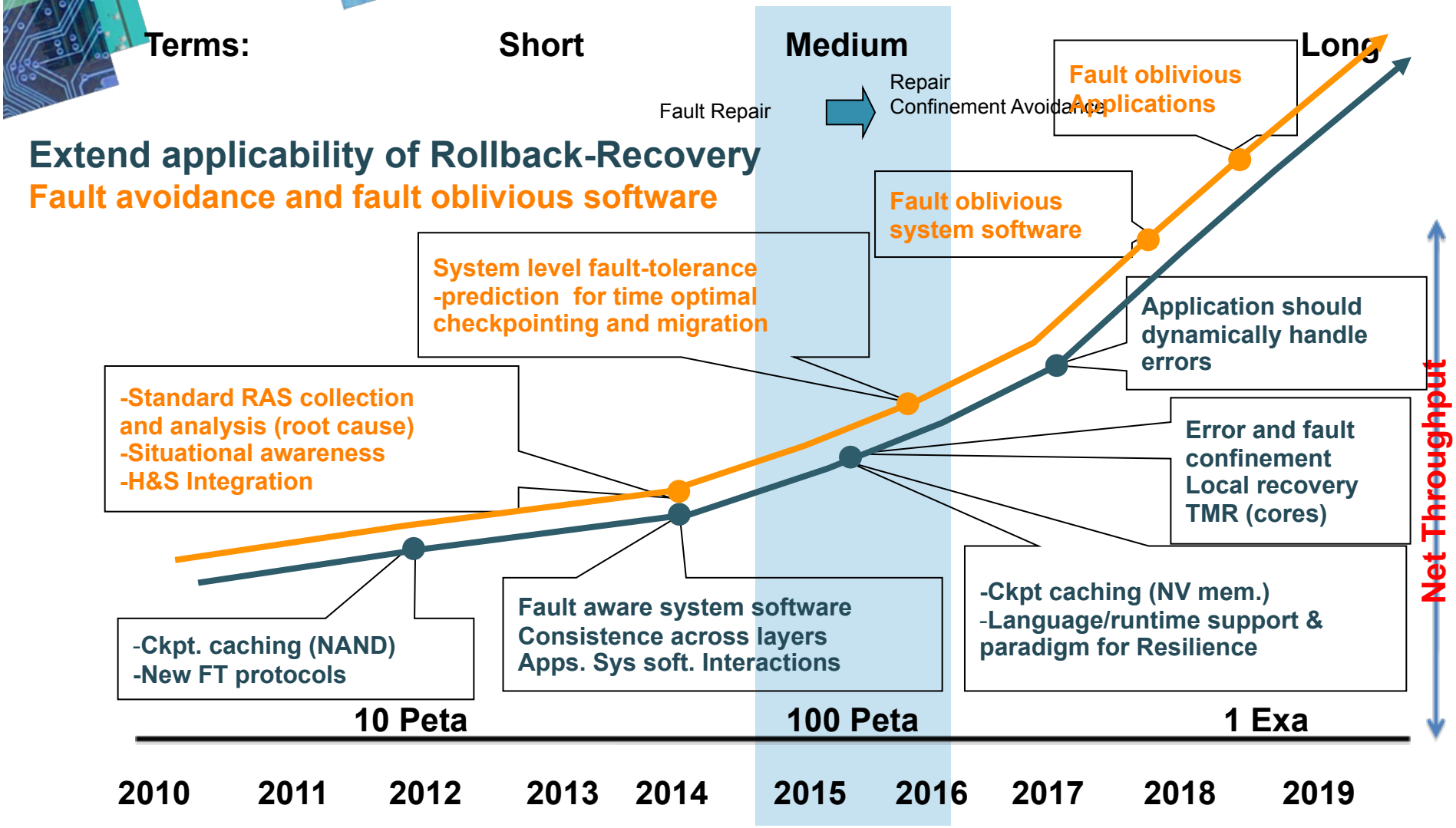
→Specific network to broadcast the checkpoint orders

→Time based coordinated checkpointing in large machines

Main issues of checkpoint-restart:

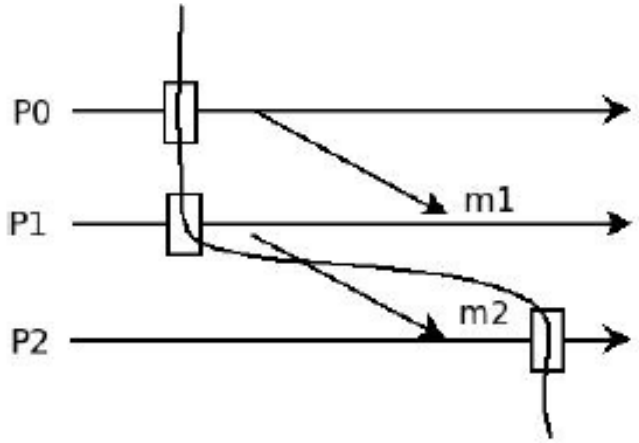
- Checkpoint are saved on file systems through the I/O nodes
- All processes are restarted even if only one fails
- In case of failure, hours of execution can be lost: checkpoint are taken based on MTBF and not just before the failures!

Exascale Resilience: IESP roadmap



Summary

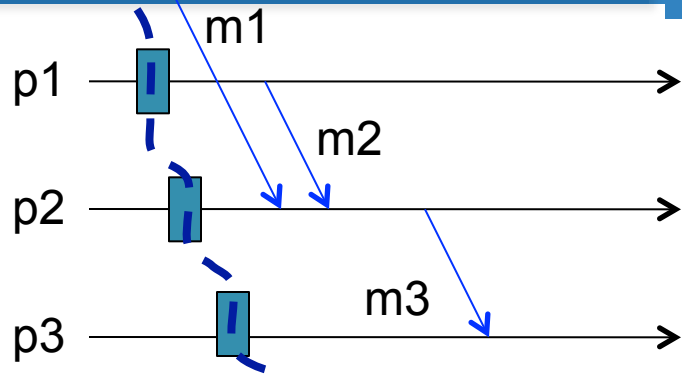
- Fault tolerance for message passing HPC applications
- **Exploring and exploiting application fundamental properties to improve fault tolerance protocols**
- Analyzing system notifications to improve fault tolerance
- Conclusion



Current Assumptions on Applications

Fault Tolerance Protocols:

-Capture the state of the execution and control the recovery to ensure that the execution after a partial or global restart is correct



The current approach uses coordinated checkpointing
It corresponds to the most generic assumptions on applications:

1) Applications may use any computation model: BSP, SPMD, Dataflow Graph, bag of tasks

2) Applications have non deterministic communication patterns

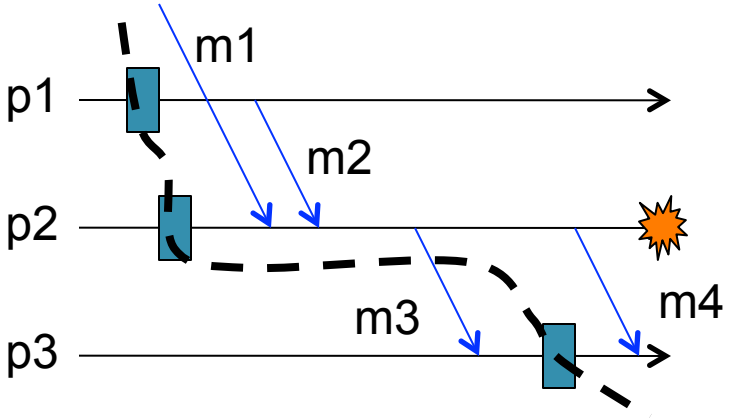
3) Applications may present any communication patterns

4) Communications and computation may be mixed in an arbitrary way

→ **Many of these assumptions are too generic for HPC applications**

Background
On fault tolerant protocols

System model:
-Fail-stop model
-A set of communicating processes
-FIFO and reliable communication channels
-Asynchronous network (no global clock)
-No I/O (for simplicity)



Main problem to address when designing fault tolerance protocols:

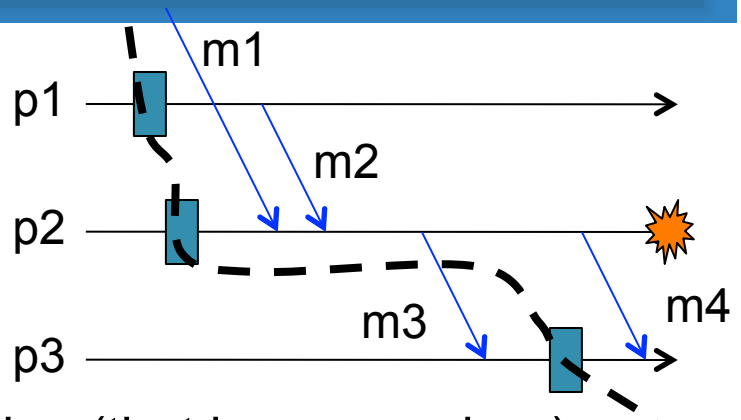
- Make sure that the execution after restart is correct: corresponds to one of the possible executions of the program (specification of prog.)
- No orphan condition (for messages)

→ No processes involved in the execution should have a state depending on a message that may not be replayed in case of restart.
→ Such situation is acceptable if you have a guarantee that the message will be preplayed the same way

→ Determinism plays an important role in the definition of FT protocols.

Main tolerant protocols & determinism

Main families of fault tolerant protocols



Uncoordinated checkpointing:
no hypothesis on determinism
no record of non deterministic events

- In case of failure, find a consistent recovery line (that has no orphan).
- Risk of domino effect: restart from the beginning

Coordinated checkpointing: no hypothesis on determinism
no record of non deterministic events

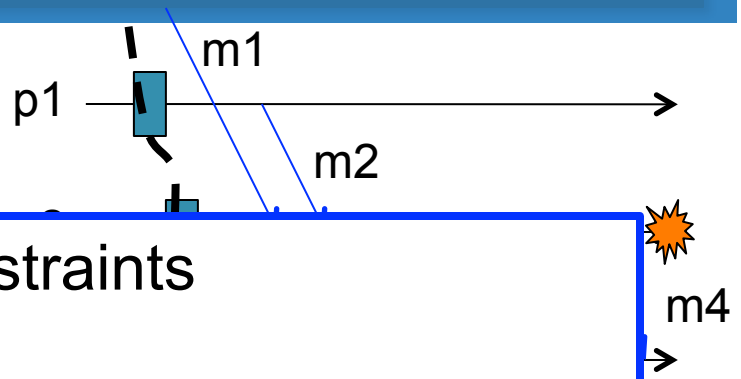
- Avoid the creation of orphan by forcing a consistent recovery line that has no orphan and by restarting all processes

Uncoordinated checkpointing with message logging: piecewise determinism
record of non deterministic events: message receptions

- All messages content and determinant (emission/reception orders) are recorded. In case of crash, logged messages will be replayed identically
- Only the failed process restarts but need to log all messages

Main tolerant protocols & determinism

Main families of fault tolerant protocols



Unco
no hy
no re
•
•
Coor
•
h

All these protocols have severe constraints

We are looking for a protocol (or a family of)
-With no risk of domino
-No need to restart all processes (restart only a small %)
-No need to log all messages (log only a small %)

Let's see what kind of determinism we have in messages passing HPC applications

Uncoordinated checkpointing with message logging: piecewise determinism
record of non deterministic events: message receptions

- All messages content and determinant (emission/reception orders) are recorded. In case of crash, logged messages will be replayed identically
- Only the failed process restarts but need to log all messages

Analyzing Fundamental Properties of HPC Applications

Joint Laboratory for Petascale Computation

INRIA NCSA

Determinism of application communication patterns (26 apps.)
INRIA-Illinois Joint lab: [ICCN2010]

Applications	Communication determinism	Applications	Communication determinism
ScaLAPACK SUMMA	Deterministic	NAS SP	Send-deterministic
NAS LU	Deterministic	NAS CG	Deterministic
NAS MG	Deterministic	NAS BT	Send-Deterministic
NAS FT	Deterministic	NAS EP	Deterministic
NAS DT	Deterministic	Nbody	Deterministic
USQCD CPS++ (MPI)	Send-Deterministic	USQCD MILK	Send-Deterministic
NERSC-6 CAM	Send-Deterministic	NERSC-6 GTC	Deterministic
NERSC-6 IMPACT	Send-Deterministic	NERSC-6 MAESTRO	Send-Deterministic
NERSC-6 PARATEC	Deterministic	SpecFEM3D	Deterministic
Jacobby	Deterministic	SQ sphot	Send-deterministic
SQ UMT	Send-Deterministic	SQ IOR	Deterministic
SQ IRS	Send-Deterministic	SQ lammps	Send-Deterministic
Ray2Mesh MW	Non-Deterministic	Ray2Mesh QD	Send-Deterministic
Master Worker	Non-Deterministic	Divide and Conquer	Send-Deterministic [1]

Table 2: Communication determinism in parallel applications

Analyzing Fundamental Properties of HPC Applications

Send determinism (INRIA-Illinois Joint lab: [ICCN2010])

```
for(i=0; i<nb_recv; i++)
   MPI_Irecv(T[i], ,i.,...);
for(i=0; i<nb_send; i++)
   MPI_Send( ..., i,...);
MPI_Waitall(nb_recv+nb_send, ...)
```

Sender rank (defined)

Receiver rank (defined)

```
if(rank ==0)
  for(i=0; i<nb_procs; i++)
   {
    MPI_Recv(T[i], ..., ANY_SOURCE, ANY_TAG,...);
   }
else MPI_Send(T[rank], ..., 0, 0, ...);
MPI_Barrier(...);
sort(T);
```

Sender rank (undefined)

Receiver rank (defined)

For a given set of input parameters, the sequence of messages sent by each process is always the same, in correct exec.

Analyzing Fundamental Properties of HPC Applications

Joint Laboratory for Petascale Computation

INRIA NCSA

Determinism of application communication patterns

INRIA-Illinois Joint lab: [ICCN2010]

Applications	Communication determinism	Applications	Communication determinism
ScaLAPACK SUMMA	Deterministic	NAS SP	Send-deterministic
NAS LU	Deterministic	NAS CG	Deterministic
NAS MG	Deterministic	NAS BT	Send-Deterministic
NAS FT	Deterministic	NAS EP	Deterministic
NAS DT	Deterministic	Nbody	Deterministic
USQCD CPS++ (MPI)	Send-Deterministic	USQCD MILK	Send-Deterministic
NERSC-6 CAM	Send-Deterministic	NERSC-6 GTC	Deterministic
NERSC-6 IMPACT	Send-Deterministic	NERSC-6 MAESTRO	Send-Deterministic
NERSC-6 PARATEC	Deterministic	SpecFEM3D	Deterministic
Jacobby	Deterministic	SQ sphot	Send-deterministic
SQ UMT	Send-Deterministic	SQ IOR	Deterministic
SQ IRS	Send-Deterministic	SQ lammps	Send-Deterministic
Ray2Mesh MW	Non-Deterministic	Ray2Mesh QD	Send-Deterministic
Master Worker	Non-Deterministic	Divide and Conquer	Send-Deterministic [1]

Table 2: Communication determinism in parallel applications

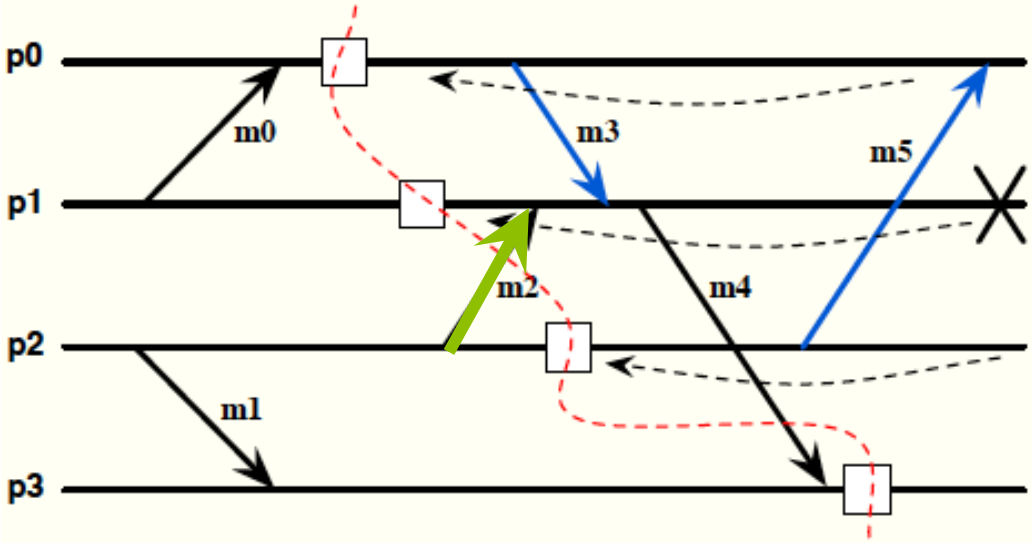
Joint Laboratory for Petascale Computation

Leveraging HPC Application properties → New fault tolerance protocols

•A first protocol INRIA-Illinois Joint lab: [IPDPS2011]

For Send-deterministic and Deterministic applications

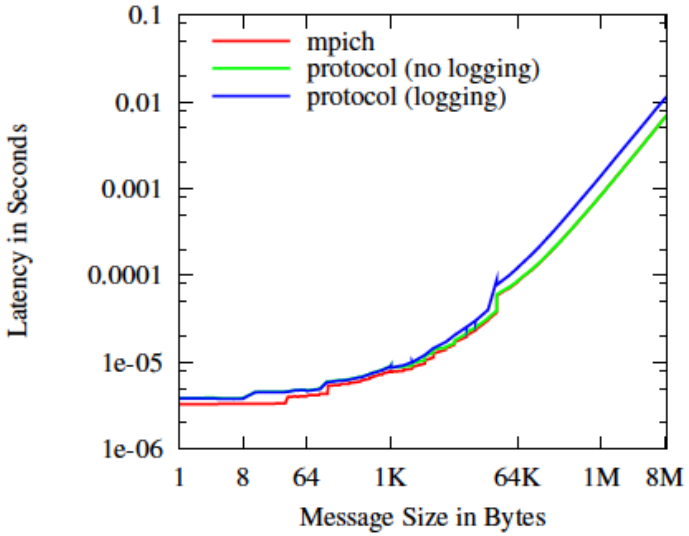
- Uncoordinated checkpointing
- No risk of inconsistency (Send-Determinism)
- Partial Message logging
- Partial reexecution
- No Domino effect

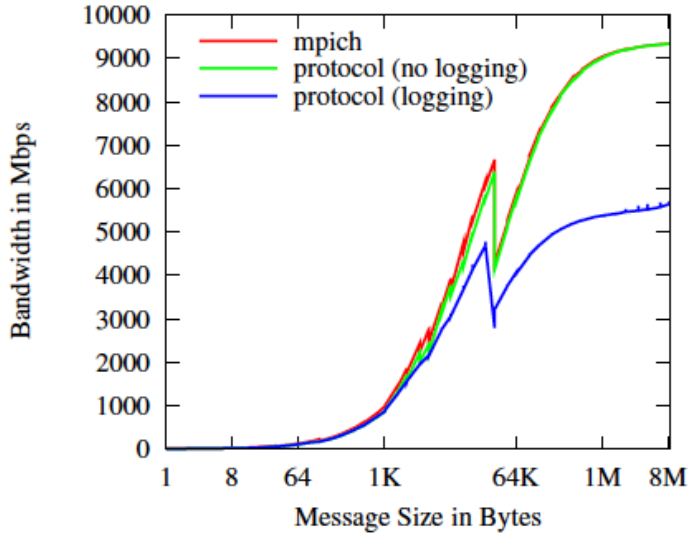


Myrinet 10G:

- Negligible impact on latency
- Memory copy for logged messages

→ ½ bandwidth



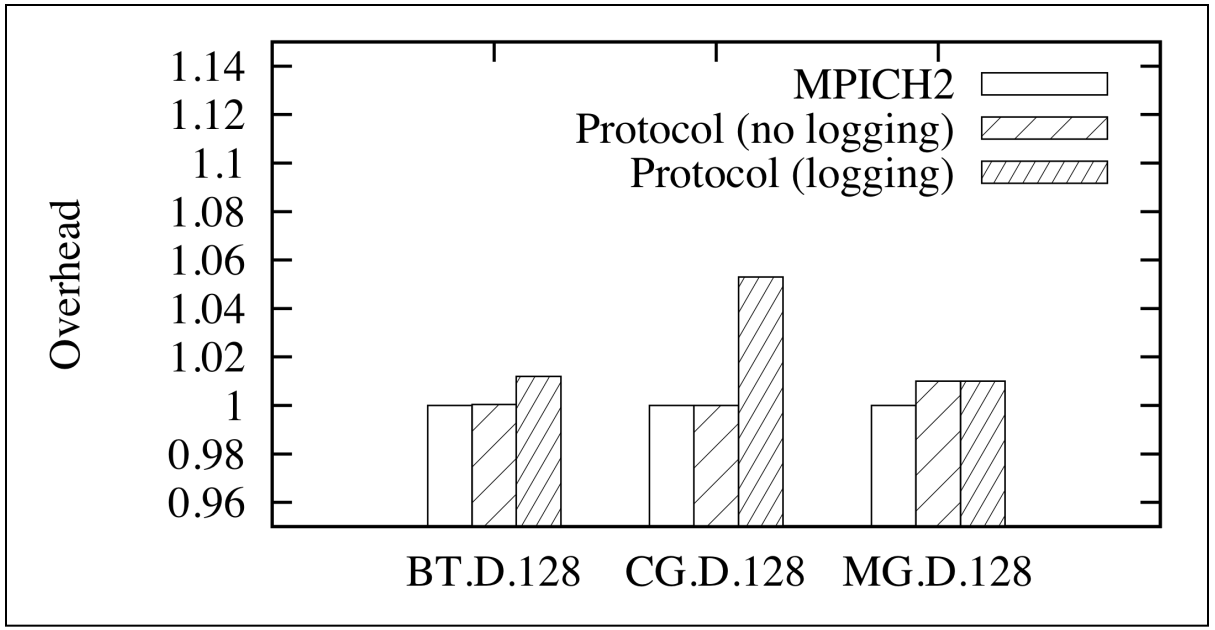


Leveraging HPC Application properties → New fault tolerance protocols

Joint Laboratory for Petascale Computation

INRIA NCSA

- A first protocol INRIA-Illinois Joint lab: [IPDPS2011]
- Uncoordinated checkpointing + Partial Message logging+ Partial reexecution+ No Domino effect



Experiment with uncoordinated checkpoints and random checkpoint time for each process:

→small number of messages need to be logged.

→But for all experiments, all processes need to roll back in case of failure

Leveraging HPC Application properties → New fault tolerance protocols

Joint Laboratory for Petascale Computation

INRIA NCSA

•A first protocol INRIA-Illinois Joint lab: [IPDPS2011]

•Uncoordinated checkpointing + Partial Message logging+ Partial

Advantage compared to existing message logging protocols:

•No need to store message determinants!

•Log only a small % of messages

Drawback:

•May require all processes to restart (1/2 with clustering)

•May require global coordination during recovery (phase)

→ not fully satisfactory

→small number of messages need to be logged.

→But for all experiments, all processes need to roll back in case of failure

Analyzing Fundamental Properties of HPC Applications

Joint Laboratory for Petascale Computation

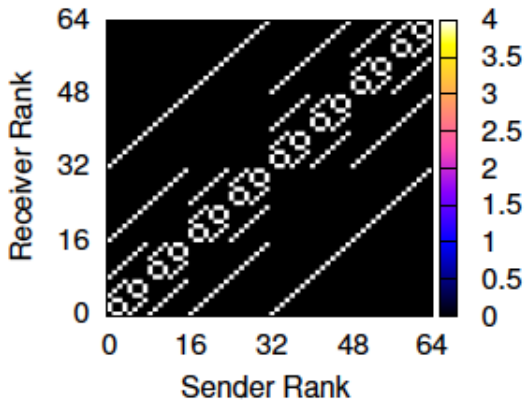
INRIA NCSA

Communication spatial locality in HPC applications:
INRIA-Illinois Joint lab: [Europar2011]

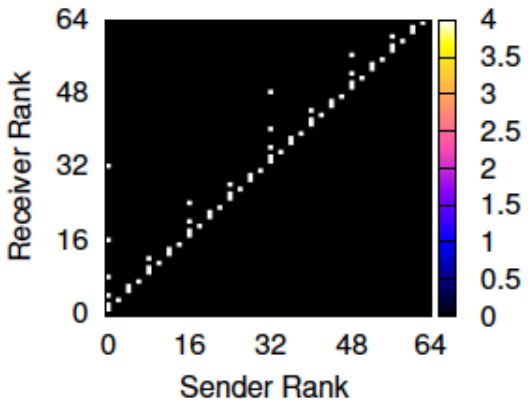
Dense linear algebra	Sparse linear algebra	Spectral methods	N-body simulations	Structured grids	Unstructured grids
BT, LU, PARATEC	CG, MAESTRO	FT, PARATEC	GTC, LAMMPS, Nbody	MG, GTC, MAESTRO, PARATEC	MAESTRO

Table 1. Dwarfs covered by the studied applications

MPI collectives

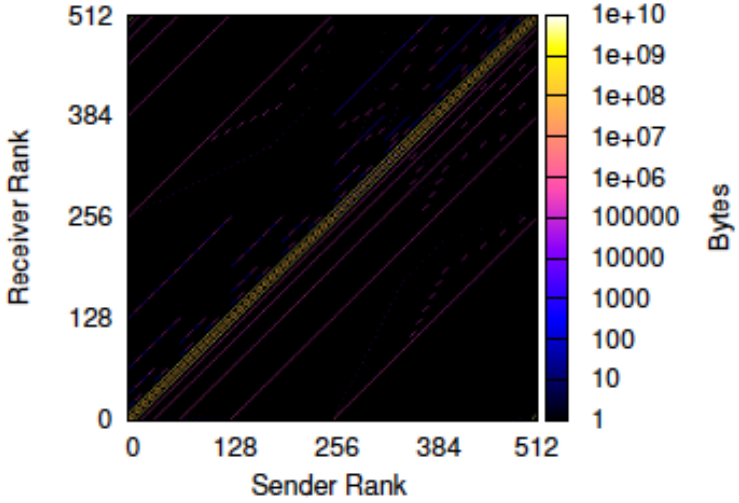


(a) Recursive Doubling



(b) Binomial Tree

GTC communications



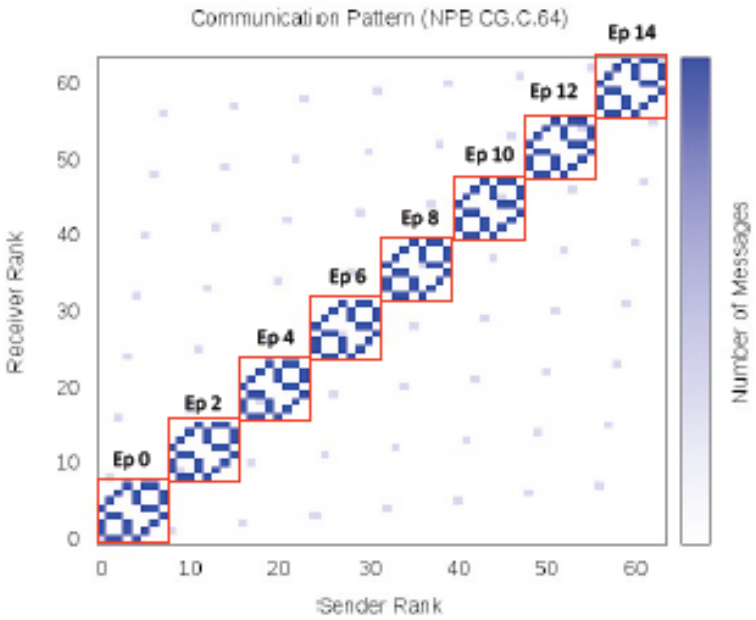
(b) GTC

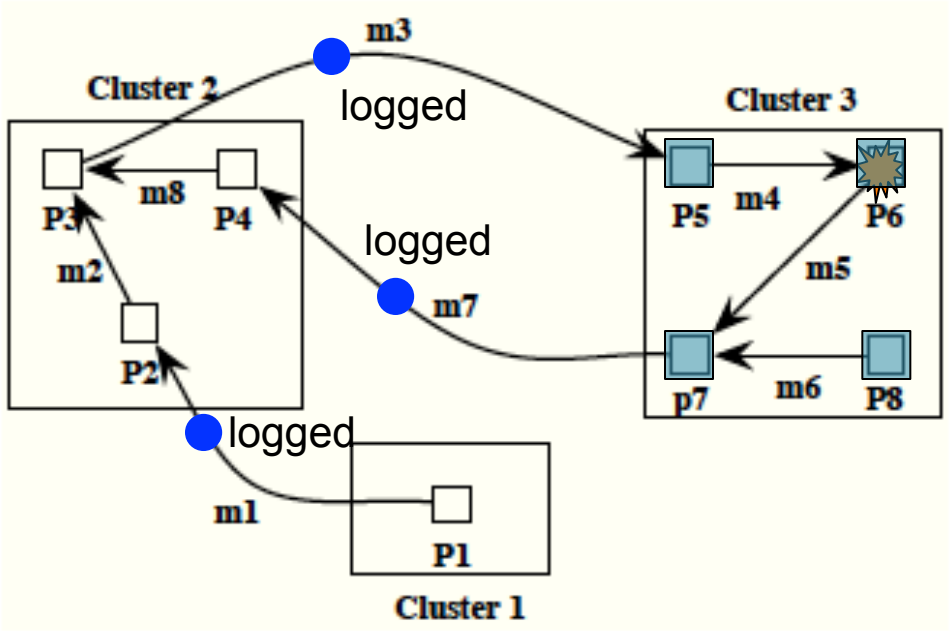
We observe in HPC applications that:
-A process communicates with a small subset of processes
-Communication patterns are highly symmetric
→ Communication patterns tends to form clusters naturally

Hybrid fault tolerant protocols

This “natural” clustering property can be used to optimize performance

But it could also be used for designing new hybrid fault tolerance protocols → coordinated ckpt. Inside clusters and message log. between





Classic hybrid fault tolerance protocols need:

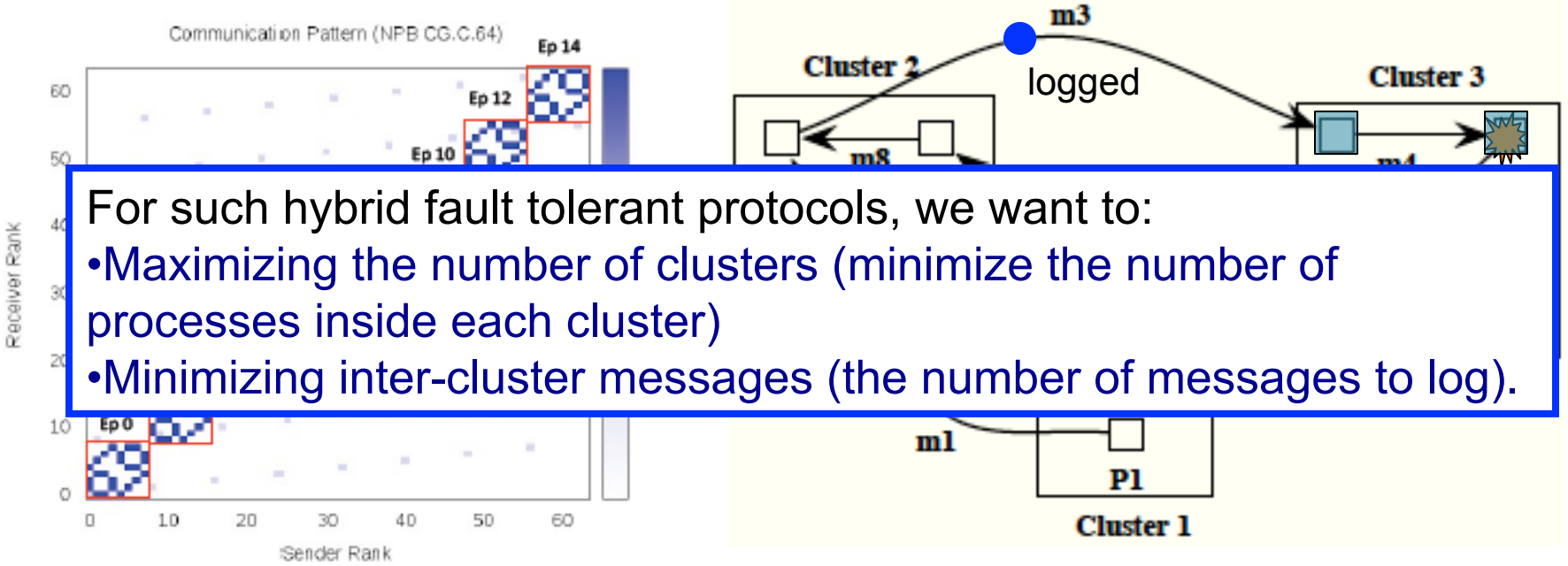
- to record the all the intra cluster message determinant
- force the replay of the intra cluster messages during recovery

→ The send determinism property allows relaxing these constaints

Hybrid fault tolerant protocols

This “natural” clustering property can be used to optimize performance

But it could also be used for designing new hybrid fault tolerance protocols → coordinated ckpt. Inside clusters and message log. between



For such hybrid fault tolerant protocols, we want to:

- Maximizing the number of clusters (minimize the number of processes inside each cluster)
- Minimizing inter-cluster messages (the number of messages to log).

Classic hybrid fault tolerance protocols need:

- to record the all the intra cluster message determinant
- force the replay of the intra cluster messages during recovery

→ The send determinism property allows relaxing these constaints

Process clustering: automatic formation of communication clusters

Joint Laboratory for Petascale Computation

Automatic cluster formation is a NP complete graph partitioning problem INRIA-Illinois Joint lab: [Europar2011]

One can use existing graph partitioning tools (MeTiS, Scotch, etc.)

They take the number of clusters to form as a parameter.

In our case, the number of clusters is a result

To address this problem, we adapted the bisection algorithm

The cost function is mixing the costs of rollback and message logging

- Most application can be clustered
- Cluster are balanced in size
- Some apps like LU achieve outstanding results (<10% roolback and logging)

Process clustering: automatic formation of communication clusters

Joint Laboratory for Petascale Computation

INRIA NCSA

Automatic cluster formation is a NP c
problem INRIA-Illinois Joint lab: [Eur
One can use existing graph partitioni
They take the number of clus
In our case, the number of clusters is

To address this problem, we adaptec
The cost function is mixing the costs

ShouldPartition: Tests if the size of the partition is higher than a threshold

Bisect: Calls one existing tool to do the bisection (PaToH).

BestSoFar: Evaluates the new partitioning based on a **cost function**.

AcceptBisection: Returns true if it improves or only slightly worsen:

- The partitioning (cost function).
- The strength of the partitioning (ratio between amount of logged messages and number of partitions)

Cost Function: defined based on the metric we want to optimize, i.e. the performance overhead. Input:

- Volume of inter-cluster messages.
- Clusters size.

- Most application can be clustered
- Cluster are balanced in size
- Some apps like LU achieve outstanding results (<10% roolback and logging)

Process clustering: automatic formation of communication clusters

Joint Laboratory for Petascale Computation

Automatic cluster formation is a NP c
problem INRIA-Illinois Joint lab: [Eur
One can use existing graph partitioni
They take the number of clus

ShouldPartition: Tests if the size of the partition is higher than a threshold

Bisect: Calls one existing tool to do the bisection (PaToH)

	Size	Nb Clusters	Min/Max cluster size	Processes to roll back	Log/Total amount of comm (in GB)	Cost
BT	1024	8	123/133	12.5%	201/1635 (12.3%)	4.37
CG	1024	32	32/32	3.1%	910/5606 (16.2%)	4.12
FT	1024	2	502/522	50%	432/864 (50%)	17.7
LU	1024	16	64/64	6.25%	67/700 (9.7%)	3.0
MG	1024	8	128/128	12.5%	20/107 (18.5%)	5.8
GTC	512	16	32/32	6.25%	240/3654 (6.6%)	2.3
MAESTRO	1024	4	252/259	25%	55/309 (17.7%)	7.17
PARATEC	1024	13	64/128	8.5%	2262/23914 (9.4%)	3.23
LAMMPS	1024	8	127/129	12.5%	0.3/4 (7.6%)	3.3
Nbody	1024	30	37/61	3.5%	80/2733 (2.9%)	1.1

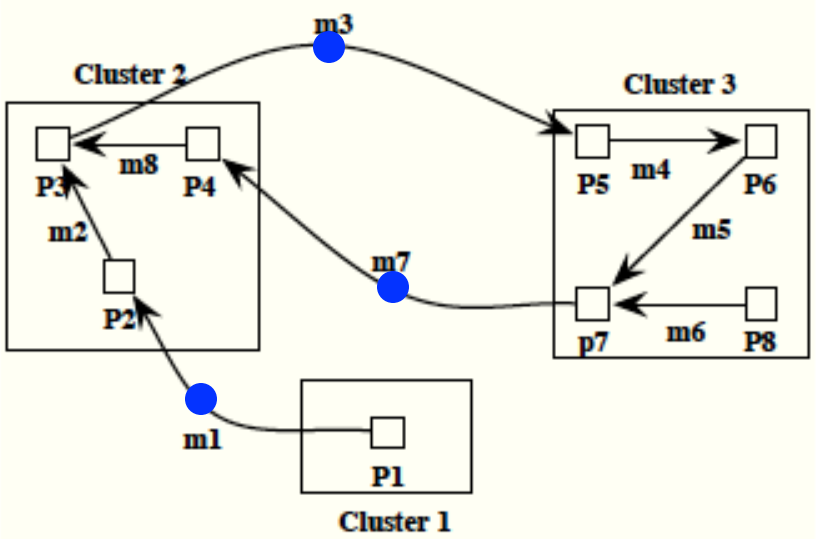
- Volume of inter-cluster messages.
- Clusters size.

•Most application can be clustered
•Cluster are balanced in size
•Some apps like LU achieve outstanding results (<10% roolback and logging)

Leveraging HPC Application properties → New fault tolerance protocols

•A second protocol INRIA-Illinois Joint lab: [IPDPS2012]

For Send-deterministic and Deterministic applications



•Hybrid protocol:
Coordinated checkpointing inside clusters
+message logging between clusters

•No risk of inconsistency (send-determinism)

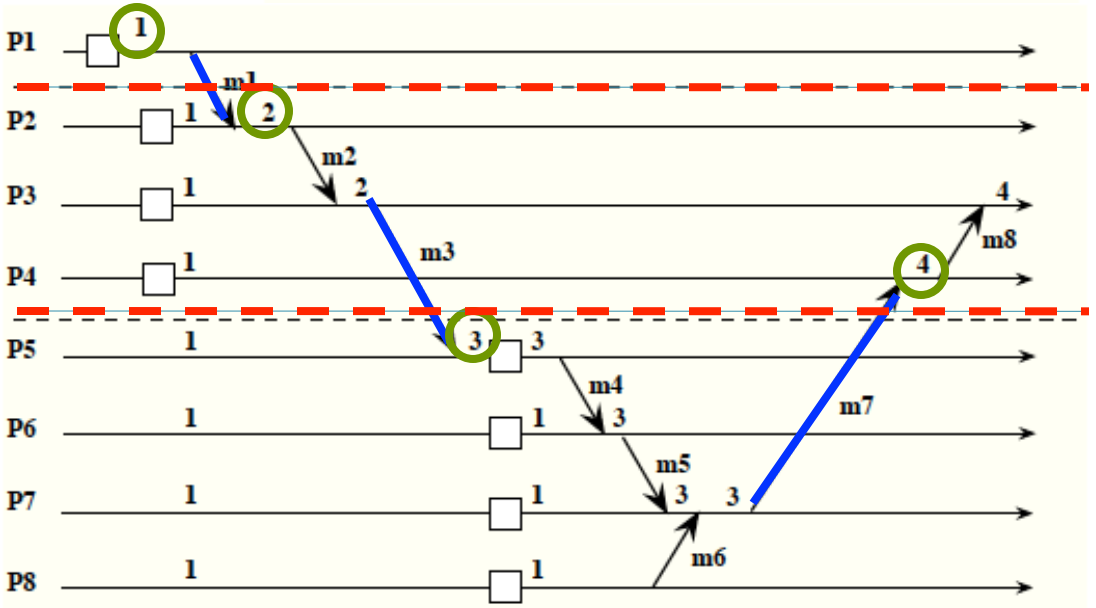
•Partial Message logging

•Content logging required for inter cluster messages (volatile memory OK)

•Keep a table of "orphans" on receiver side for filtering and managing causalities at restart: to avoid confusion (m2§m8 on P3)

•Partial reexecution

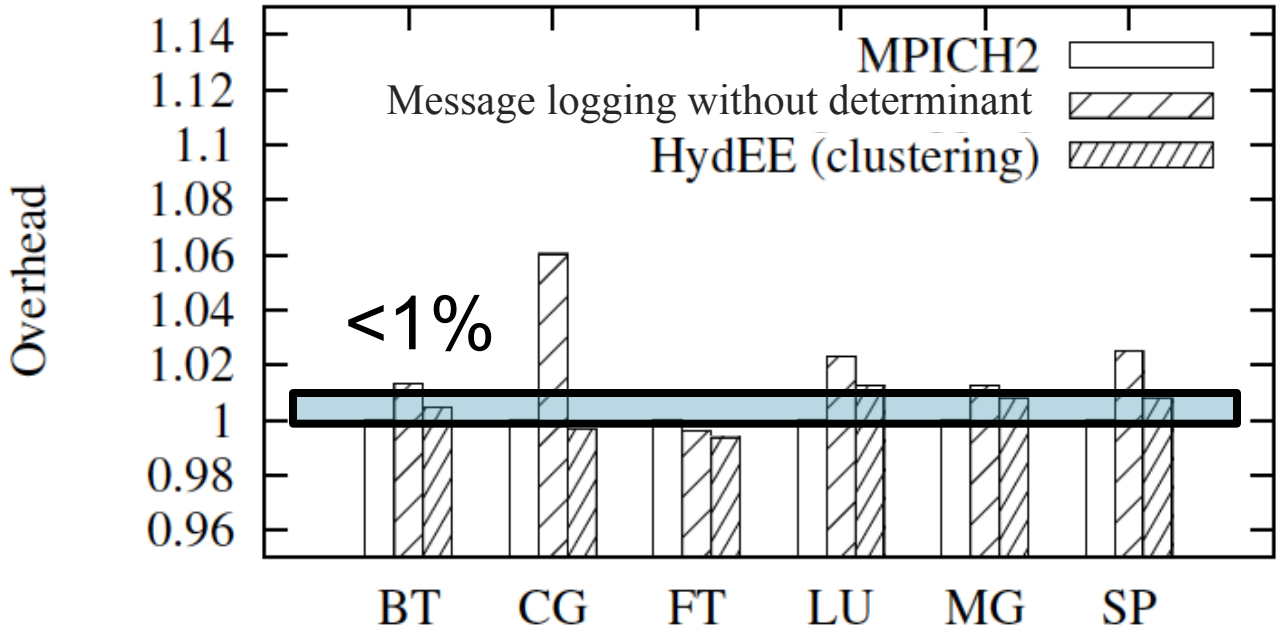
•No Domino effect



Joint Laboratory for Petascale Computation

Leveraging HPC Application properties → New fault tolerance protocols

- A second protocol INRIA-Illinois Joint lab: [IPDPS2012]
- Hybrid protocol: Coordinated checkpointing inside clusters +message logging between clusters



	Nb Clusters	Process 0 cluster size	Processes to roll back	Log/Total Amount of data (in GB)
NPB BT	5	63	21.78%	143/791 (18.09%)
NPB CG	16	16	6.25%	440/2318 (18.98%)
NPB FT	2	129	50%	431/860 (50.19%)
NPB LU	8	32	12.5%	44/337 (13.26%)
NPB MG	4	64	25%	13/66 (19.63%)
NPB SP	6	32	18.56%	289/1446 (20.04%)

Joint Laboratory for Petascale Computation

Leveraging HPC Application properties → New fault tolerance protocols

INRIA NCSA

- A second protocol INRIA-Illinois Joint lab: [IPDPS2012]
- Hybrid protocol: Coordinated checkpointing inside clusters

Advantage compared to flat protocols:

- Does not require to restart all processes
- Coordination is confined inside cluster
- Does not need to log all messages

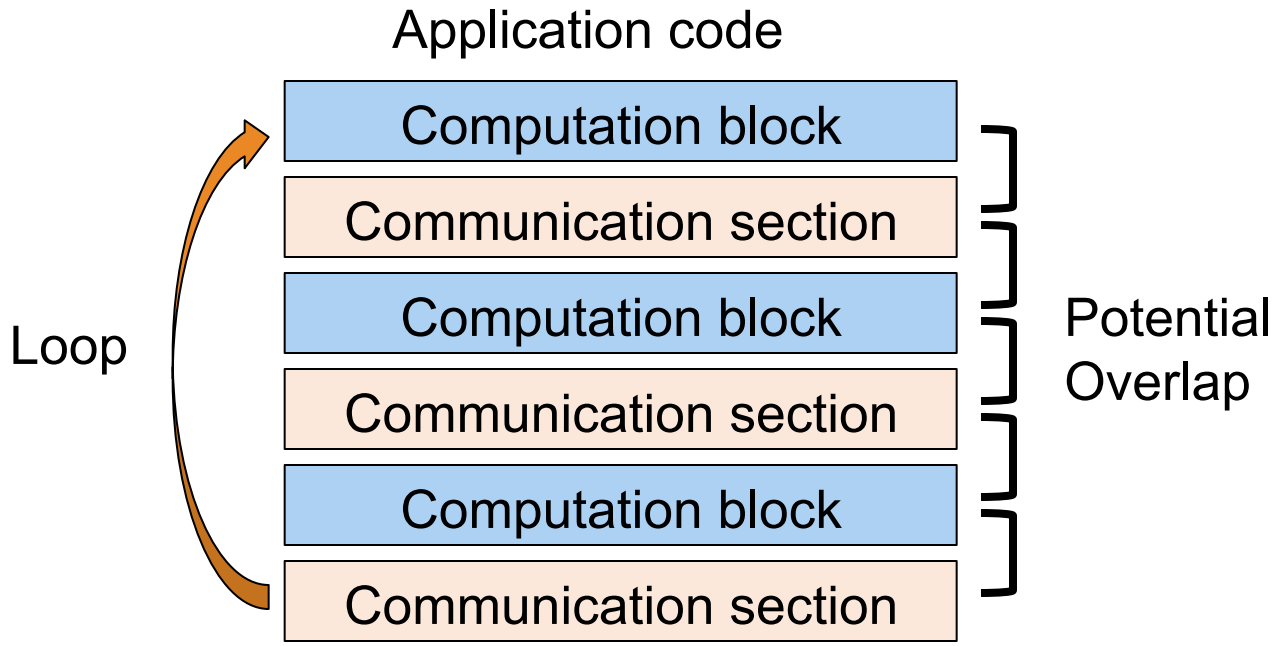
Advantage compared to existing hybrid protocols (send-determinism):

- Correct!
- Does not require to log determinant of intra cluster messages on stable storage

However, recovery needs coordination and cannot be fully decentralized.

NPB				
NPB				
NPB				
NPB				
NPB				
NPB SP	6	32	18.56%	289/1446 (20.04%)

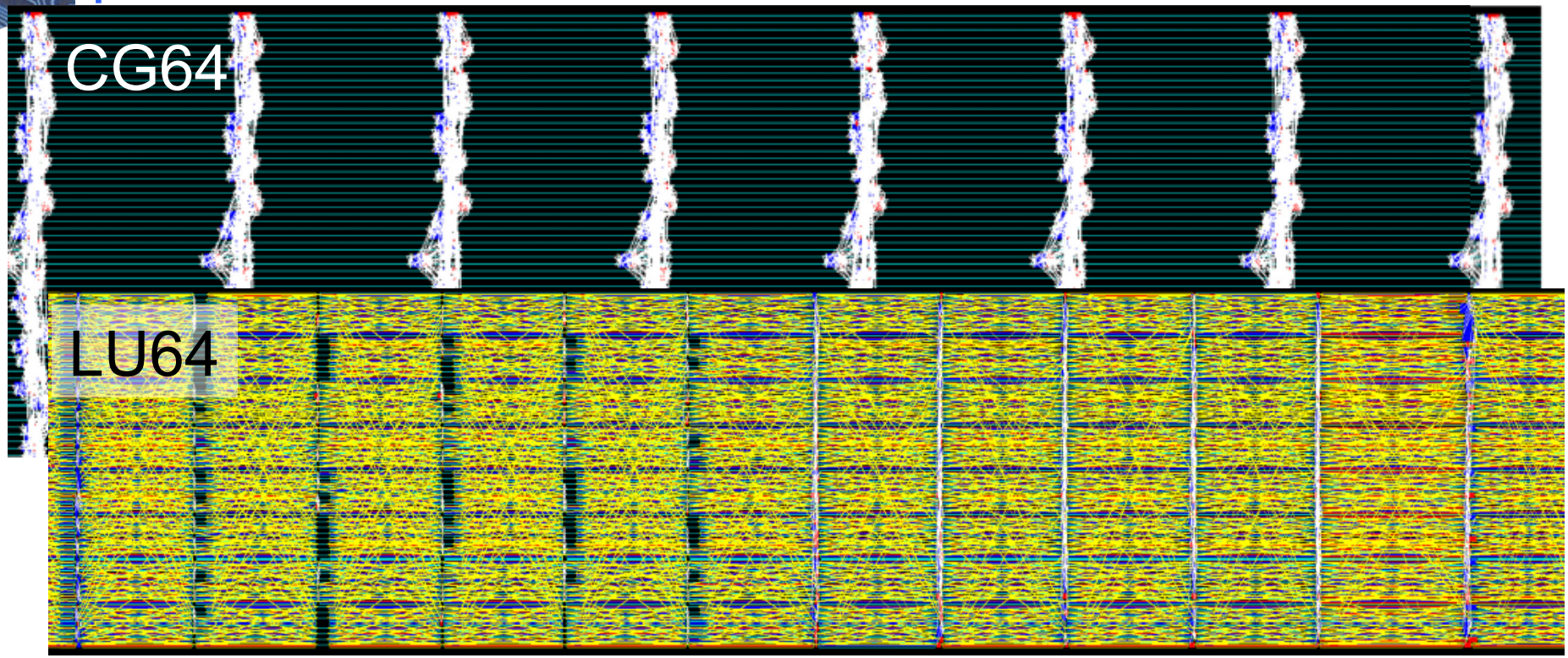
Alternation of computation and communication phases



The separation between communication patterns could be explicit (barrier between communication patterns) or implicit (local wait at the end of a communication pattern)

Analyzing Fundamental Properties of HPC Applications

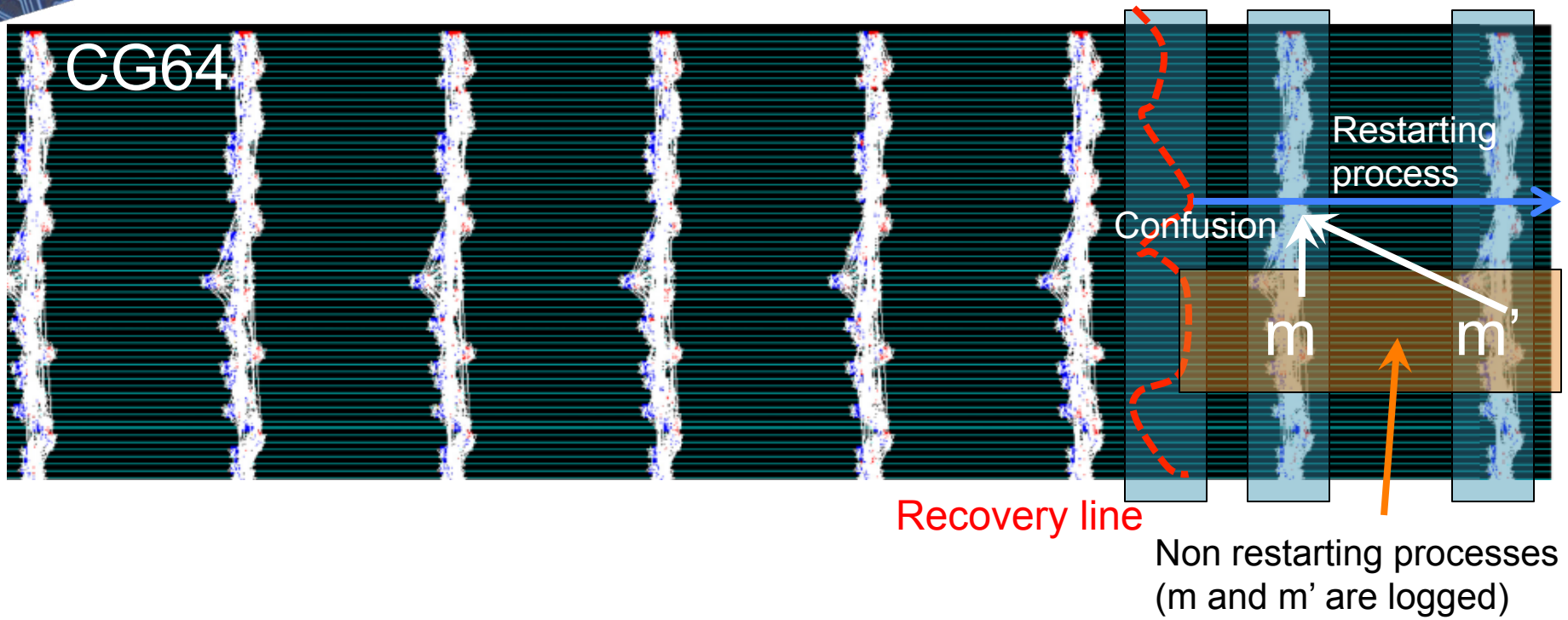
Alternation of computation and communication phases



The separation between communication patterns could be explicit (barrier between communication patterns) or implicit (local wait at the end of a communication pattern)

Notion of communication determinism

Recovery is complex for both new protocols because we need to avoid confusion between comm. sections.



To avoid confusion, application processes would need to feature the send-determinism property during fault free execution AND recovery

However this is not the case « naturally » → Codes need to be adapted

Joint Laboratory for Petascale Computation

Example of confusion

INRIA NCSA

Here is a sequence of 2 confusing patterns

Process n

```
for(i=0; i<nb; i++) {
    MPI_Irecv(T[i],…,ANY_SOURCE,…);
/* where source is unknown*/
    MPI_Isend(x, ..., i, ...);
/* where i is the destination */
}
MPI_WaitAll(...);
Sort(T);
. . .
```

Process m

```
for(i=0; i<nb; i++) {
    MPI_Irecv(T[i],…,ANY_SOURCE,…);
/* where source is unknown*/
    MPI_Isend(x, ..., i, ...);
/* where i is the destination */
}
MPI_WaitAll(...);
Sort(T);
. . .
```

Logged message

Correct matching

```
for(i=0; i<nb; i++) {
    MPI_Irecv(T[i],…,ANY_SOURCE,…);
/* where source is unknown*/
    MPI_Isend(x, ..., i, ...);
/* where i is the destination */
}
MPI_WaitAll(...);
Sort(T);
```

```
for(i=0; i<nb; i++) {
    MPI_Irecv(T[i],…,ANY_SOURCE,…);
/* where source is unknown*/
    MPI_Isend(x, ..., i, ...);
/* where i is the destination */
}
MPI_WaitAll(...);
Sort(T);
```

Logged message

Example of confusion

Here is a sequence of 2 confusing patterns

Restarting process

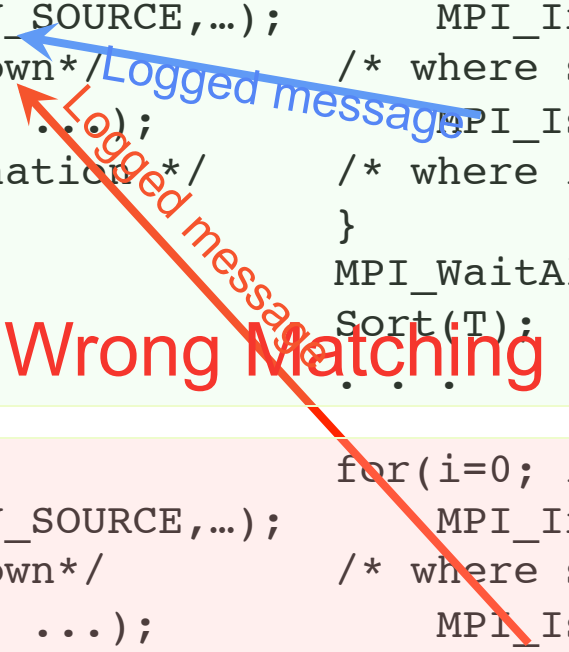
```
for(i=0; i<nb; i++) {
    MPI_Irecv(T[i],…,ANY_SOURCE,…);
/* where source is unknown*/
    MPI_Isend(x, ..., i, ...);
/* where i is the destination */
}
MPI_WaitAll(...);
Sort(T);
. . .
```

```
for(i=0; i<nb; i++) {
    MPI_Irecv(T[i],…,ANY_SOURCE,…);
/* where source is unknown*/
    MPI_Isend(x, ..., i, ...);
/* where i is the destination */
}
MPI_WaitAll(...);
Sort(T);
```

Non restarting process

```
for(i=0; i<nb; i++) {
    MPI_Irecv(T[i],…,ANY_SOURCE,…);
/* where source is unknown*/
    MPI_Isend(x, ..., i, ...);
/* where i is the destination */
}
MPI_WaitAll(...);
Sort(T);
. . .
```

```
for(i=0; i<nb; i++) {
    MPI_Irecv(T[i],…,ANY_SOURCE,…);
/* where source is unknown*/
    MPI_Isend(x, ..., i, ...);
/* where i is the destination */
}
MPI_WaitAll(...);
Sort(T);
```



Code adaptation to avoid confusion

Differentiate the two communication patterns

Restarting process

```
for(i=0; i<nb; i++) {
  MPI_Irecv(T[i],…,ANY_SOURCE,TAGi);
/* where source is unknown*/
  MPI_I
/* where
}
MPI_Wai
Sort(T)
…

for(i=0
  MPI_I
/* where
  MPI_I
/* where
}
MPI_WaitAll(...);
Sort(T);
```

Non restarting process

```
for(i=0; i<nb; i++) {
  MPI_Irecv(T[i],…,ANY_SOURCE,TAGi);
/* where source is unknown*/
                                    /
                                    AGk);
                                    /
}
MPI_WaitAll(...);
Sort(T);
```

Advantage compared to existing protocols for send-deterministic applications:

- No need to manage causalities
- Much simpler and fully distributed recovery process

Drawback:

Developers must adapt application codes

No confusion even during partial restart: keep send determinism

Wrapping up part 1

We have presented 3 fundamental properties of message passing HPC applications

1) Determinism of communications

   -allows avoiding the determinant storage overhead

2) Spatial locality of communication patterns

   -allows process clustering, partial restart and partial message logging (content)

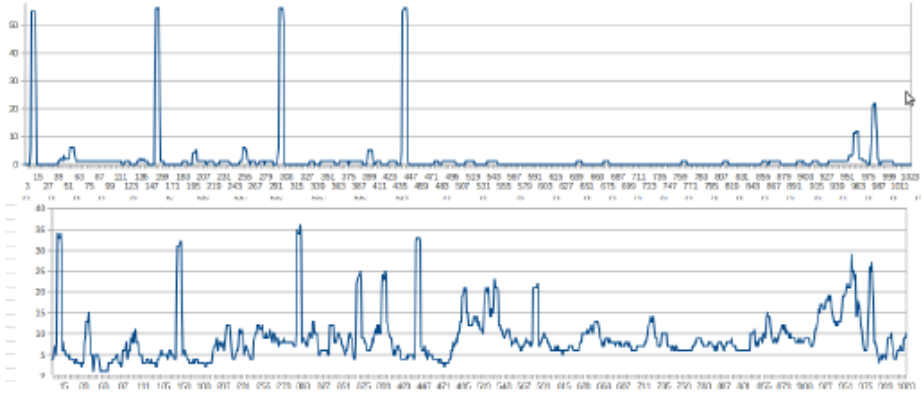
3) Separated communication patterns

   -allows fully distributed restart (requires code adaptation)

→ New FT protocols: no determinant storage, no domino effect, partial restart, partial message logging and fully distributed restart

Summary

- Fault tolerance for message passing HPC applications
- Exploring and exploiting application fundamental properties to improve fault tolerance protocols
- **Analyzing system notifications to improve fault tolerance**
- Conclusion



Joint Laboratory for Petascale Computation

Event Log analysis: Using Signal analysis Approach

INRIA NCSA

HPC system generates a lot of events about hardware conditions, software activities, etc.

Examples:

Header	Message
[02:32:47][c1-0c1s5n0]	Added 8 subnets and 4 addresses to DB
[02:32:51][c3-0c0s2n2]	address parity check..0
[02:32:52][c3-0c0s2n2]	address parity check..1
[02:32:57][c1-0c1s5n0]	Added 10 subnets and 8 addresses to DB
[02:34:21][c2-0c1s4n1]	data TLB error interrupt

Time stamp Location

These events are stored in log files.

We can analyze log files for:

- Optimal checkpoint interval computation
- Detection of abnormal behaviors, Root cause analysis, Event prediction
  Predict errors would allow proactive checkpointing, migration

The general approach for event log analysis is to use data mining algorithms to group events of same type, extract event correlations
First step is event Clustering

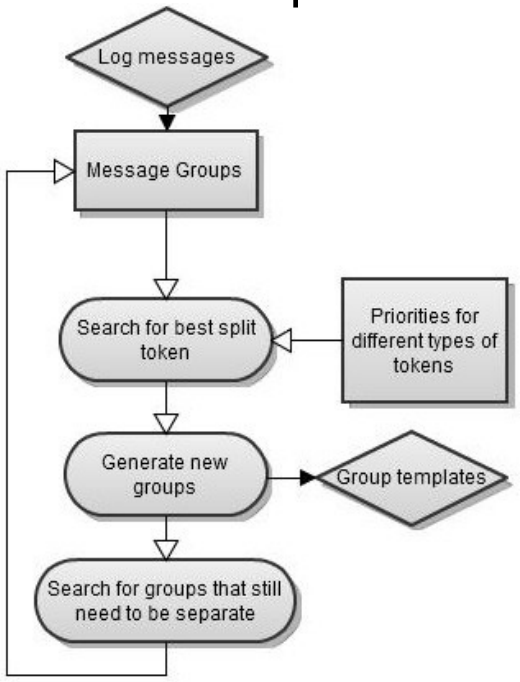
Clustering events (divide events in groups)

HELO: Hierarchical Event Log Organizer [Europar2011]

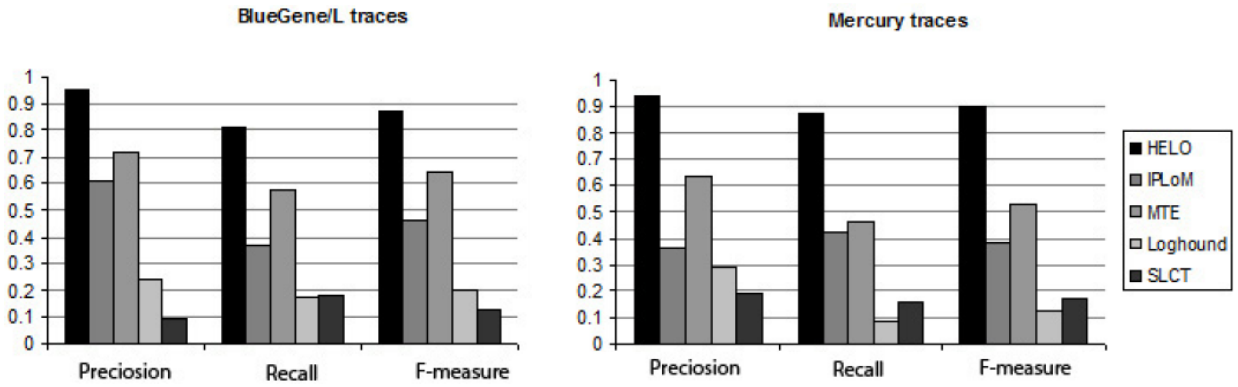
- HELO: unsupervised clustering leveraging semantic aspects of event logs
- Events of the same type have the same line format (length may differ)
- HELO also generates templates for event groups that would be used for event correlation analysis

Example of event: [2008-07-08 02:32:47][c1-0c1s5n0] ***157 CMC Errors***

Recursive split:



*Example of Template : node card * check: missing * node*



Pre: among the generated clusters, what is the proportion of the ones that a sys admin would generate (correctness)

Rec: among all the clusters that a sys admin would generate, what is the proportion of clusters that HELO generates (completeness)

Loghound, SLCT (Apriori), IPMLoM (3 phases), MTE (const. + Variables)

Optimal checkpoint interval

Per component approach [SC2011]

- State of the practice: All syst. components + Exponential dist. → Young form.
- However, applications may not use all components of the system
- We made a per component failure characterization using HELO on Mercury AND computed the check. interval based on components actually used

The main conclusions from the Mercury log analysus are:

1) Best fitting distributions is not exp. (except. for full small executions)
2) We can reduce the checkpoint frequency (overhead) by computing the checkpoint interval from the used components

Optimal TC Checkpoint Intervals (hours)

		$M_J = 8$	$M_J = 16$	$M_J = 32$	$M_J = 64$	$M_J = 128$	$M_J = 256$	$M_J = 512$
F_{all}	T_S =1 min	3.746	2.912	2.236	1.669	1.222	0.923	0.681
	T_S =10 min	11.77	9.136	6.995	5.218	3.810	2.881	2.134
	T_S =30 min	20.24	15.68	11.98	8.923	6.495	4.906	3.640
F_5, F_6 Mostly CPU+Mem	T_S =1 min	18.99	13.38	9.430	6.713	4.594	3.327	2.328
	T_S =10 min	60.17	42.42	29.90	21.30	14.57	10.56	7.392
	T_S =30 min	104.55	73.67	51.91	36.97	25.25	18.32	12.81
F_1, F_5, F_6 Mostly CPU+Mem+disc	T_S =1 min	10.44	7.345	5.171	3.619	2.547	1.831	1.253
	T_S =10 min	33.16	23.24	16.28	11.37	7.972	5.718	3.882
	T_S =30 min	57.69	40.27	28.06	19.55	13.66	9.767	6.579

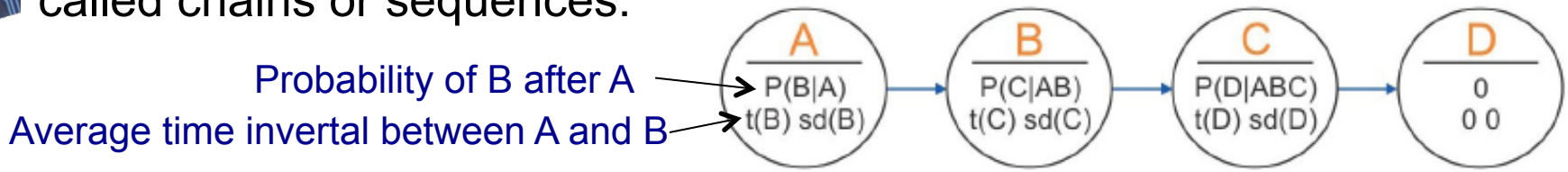
Factor of ~5 (when k~1: Exponential)

Factor of ~4

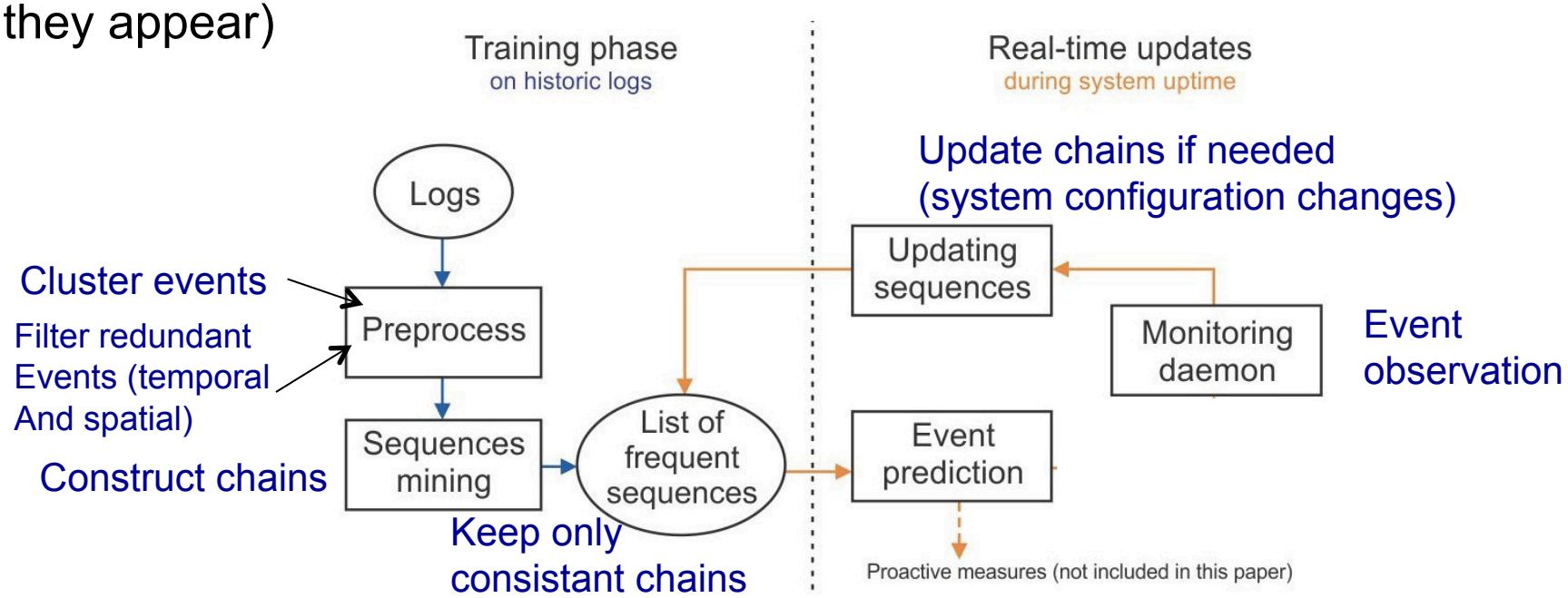
Event prediction

Dynamic Time Window [SLAMS2011]

Event prediction is based on association rules extracted from event logs called chains or sequences:

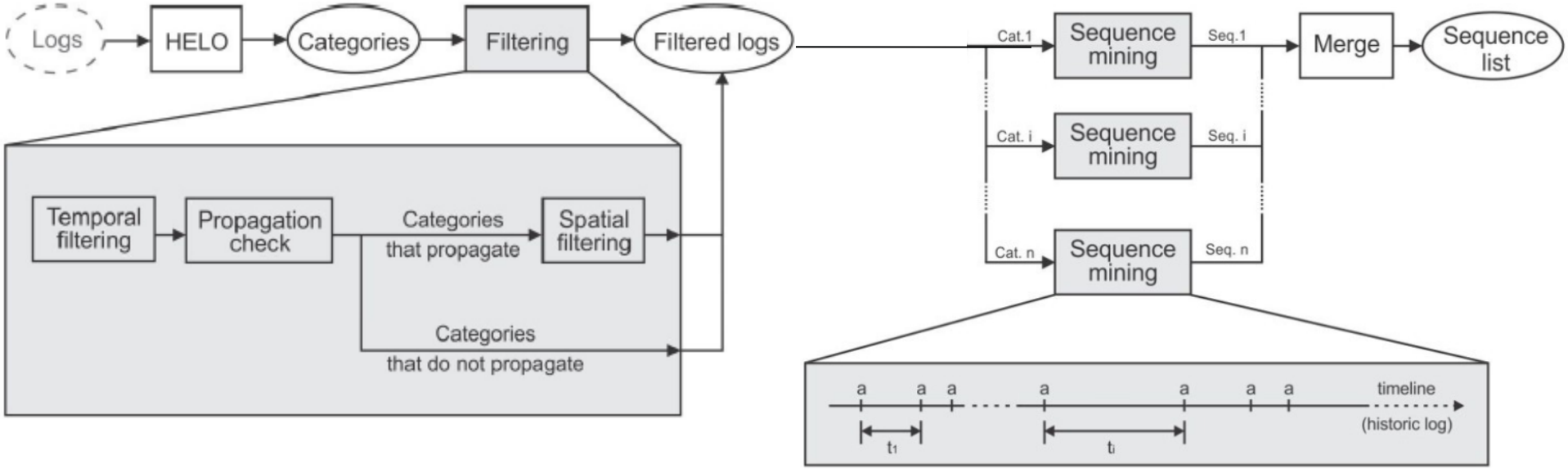


To predict, we first need to learn association rules offline (training period) and then observe the system online (and follow chains when they appear)



Adaptive event prediction

Dynamic Time Window [SLAMS2011]

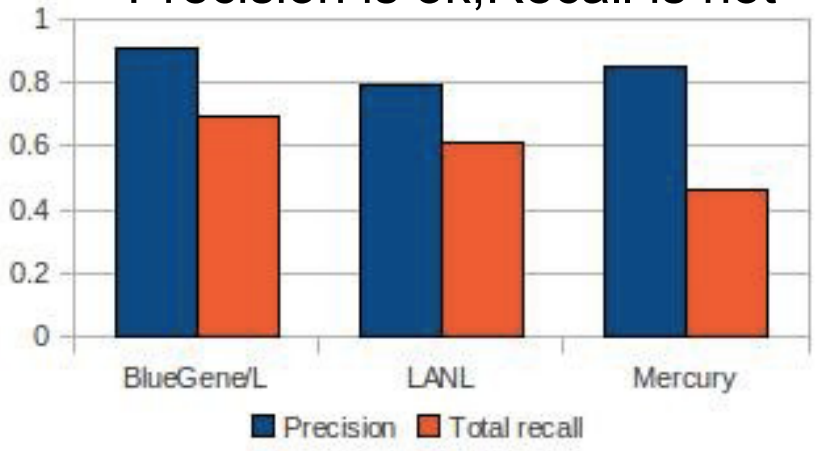


Precision: percentage of correctly predicted events over all predicted events

Recall: percentage of predicted events over all events

System	Mercury	BlueGene/L	LANL
Number of nodes	635	5252	256
Number of msg	100G	10G	433K
Initial templates	215	171	30
Templates after update	321	207	52
Analyse interval	Jan 2008 to July 2008	June 2005 to Dec 2006	2003 to 2006

Precision is ok,Recall is not



Using signal analysis for log analysis

[IPDPS2012]

Any signal variation would reflects a deviation from normality seen as a potential precursor of failure

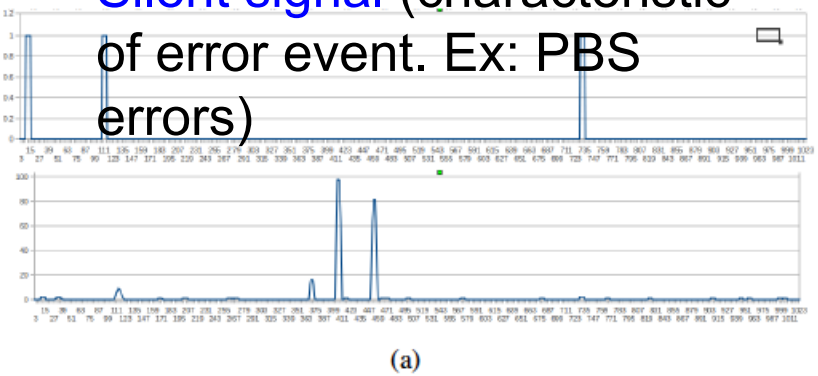
Signals extraction:

- Select a group of events
- Use a sampling rate (different for each group of events)
- Map number of events for each sample

→Time series for each event group

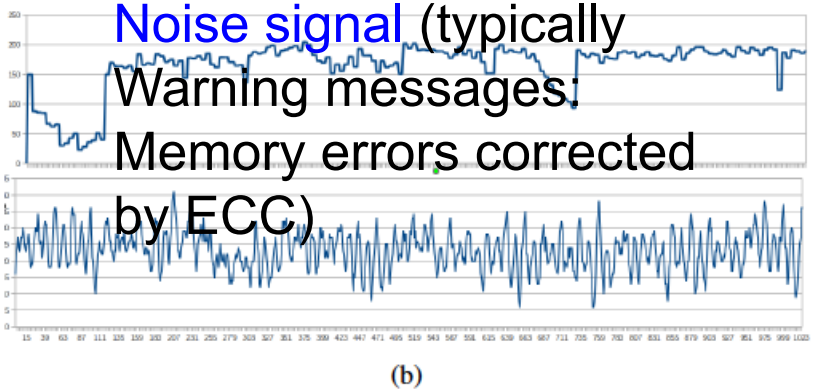
→ Interpreted as Signals:

- Frequency
- Intensity



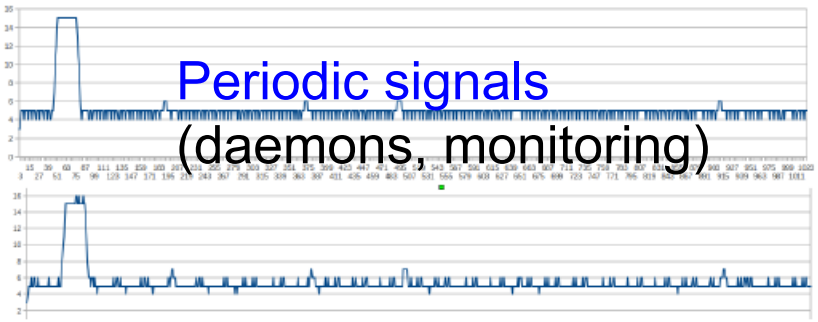
Silent signal (characteristic of error event. Ex: PBS errors)

(a)



Noise signal (typically Warning messages: Memory errors corrected by ECC)

(b)



Periodic signals (daemons, monitoring)

(c)

Signals analysis workflow

Start by considering the signal as periodic

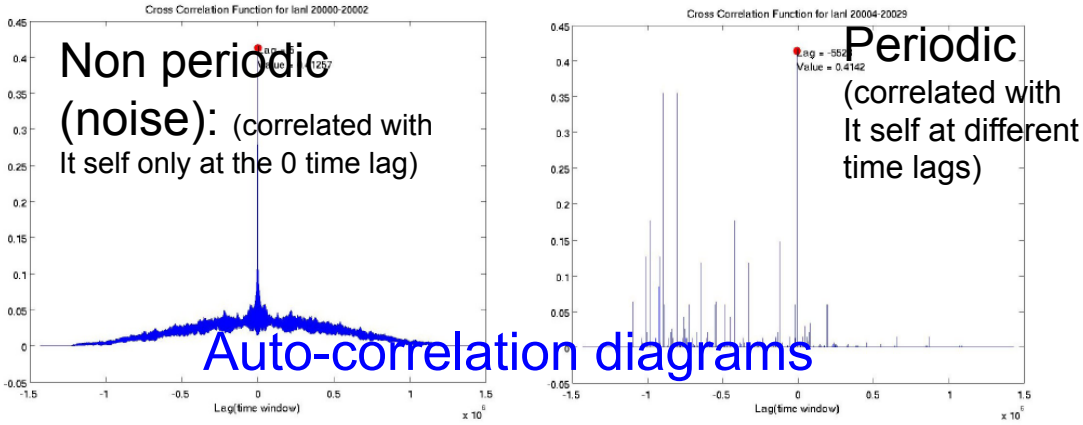
1) Establish the best sampling rate: Nyquist Theorem

-start with a low sampling rate (min time between 2 occurrences of the event type) and increase the rate until it exceeds the minimum time lag between two events of same type OR a periodic signal if found

For each tested sampling rate:

Determine if the extracted signal is periodic: auto-correlation function

Compute the similarity between a signal and itself for different time lag: if similarity over a threshold, signal is periodic



Non periodic (noise): (correlated with It self only at the 0 time lag)

Periodic (correlated with It self at different time lags)

Auto-correlation diagrams

2) If periodic, Compute the power spectrum of the signal from FFT

-Filter the signal to remove noise (to keep the strongest harmonics)

Signals analysis workflow

2) If not periodic, treat the signal as noise or silent

-Use a fixed sampling rate (10s): Since no period found, we need to use sampling rate tractable by the following signal analysis tools (size should allow fast analysis)

3) Classify signal as noise or silent based on the majority behavior

-Silent signal is “silent” for the majority of the sampling interval. Noise is opposite.

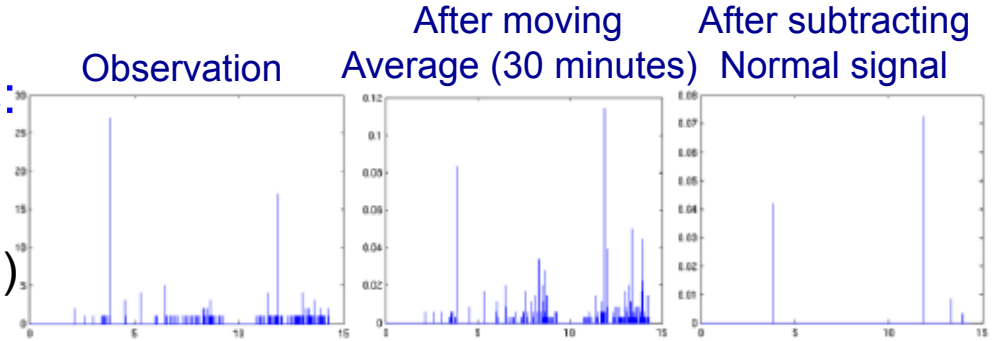
System	Mercury	LANL
Periodic signals		
Number	11	2
Percentage	2.7%	3.8%
Correct frequencies	90.9%	100%
Silent signals		
Number	338	39
Percentage	82.6%	73.6%
Noise signals		
Number	60	12
Percentage	14.7%	22.6%

Anomaly Detection

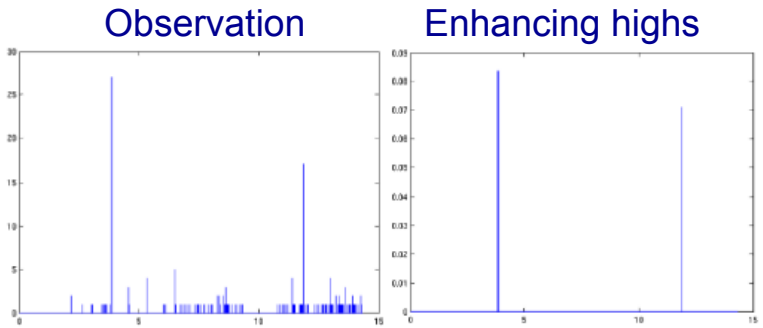
Basically we want to detect changes in the structure of the “normal” signal
Change of frequency or intensity could indicate an anomaly

For the 3 kinds of signals, we observe that changes are either a decreasing time interval between events or a drop to 0 intensity (no reduction of time interval).

Detect changes in frequency at which the system generates events:
-use the moving average technique (replace each value with an average of neighboring values: highlight high rates)
-filter the resulting signal to remove the normal behavior



Detect changes in intensity (changes in the total number of events generated by the system per time unit):
-2 filters (to enhance differences between outliers and and normal signal)

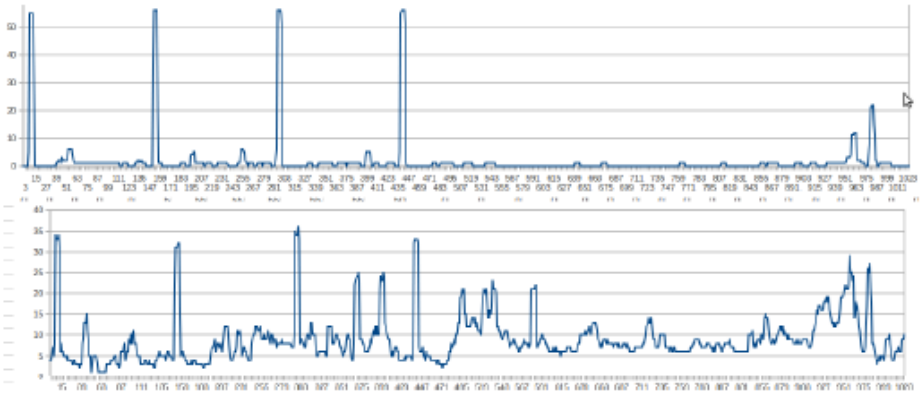


Event correlation

We want to detect correlations between anomalies (not raw signals):

-use cross correlation functions on processed signals

(measure of similarity of two waveforms as a function of a time-lag applied to one of them.)



On raw signals, almost no correlation found:

-normal behavior of the signals may be different (similarities are masked)

-cross correlation function is not associative (so order matters and there are too may combinations)

The signal processing techniques previously presented, allow focusing on anomaly correlations

System	Mercury	LANL
Average lag (seconds)	45	23
Correlated events (percentage)	68%	71%

Failure prediction

More complex than event predictions (predict problems not normal events)

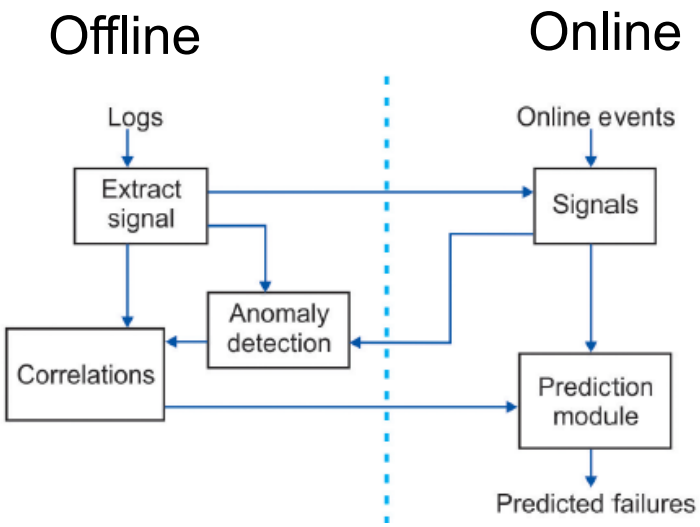
We use 1/10 of the LANL log for training and prediction is done on the rest

We compare the anomaly prediction from event log with the actual failures log (LANL only because there is a failures log in addition to an event log)

Distribution of failure types in LANL logs

Facilities	Hardware	Human Error
1.55%	61.57%	0.64%
Network Error	Software	Unknown
1.8%	23.02%	11.38%

Prediction methodology



Prediction results

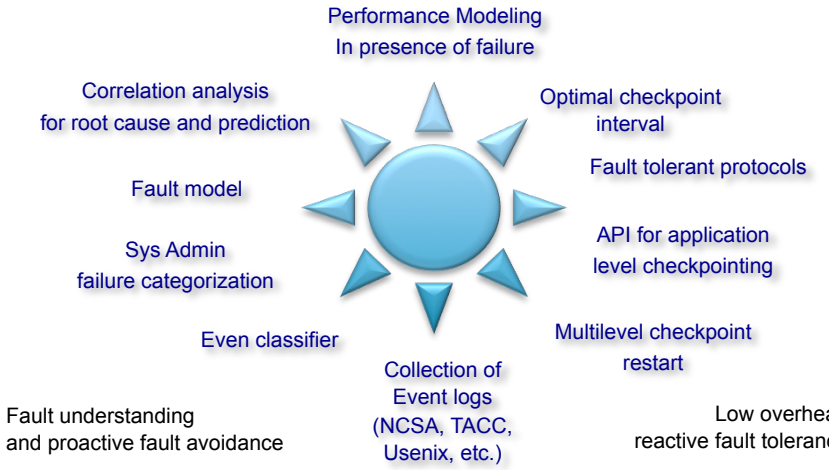
Precision	93%
Recall	43%
Average lag (before)	32 seconds

	Facilities	Hardware	Human Error
Precision	89.2%	93.8%	80.8%
Recall	38%	45.1%	9.2%
Average lag	68s	45s	15s

	Network Error	Software	Unknown
Precision	91.2%	93.7%	91.6%
Recall	42.8%	41.1%	23.4%
Average lag	48s	5s	19s

Summary

- Fault tolerance for message passing HPC applications
- Exploring and exploiting application fundamental properties to improve fault tolerance protocols
- Analyzing system notifications to improve fault tolerance
- **Conclusion**



Wrapping-up

What you may want to take home from this talk:

1) HPC applications have some nice properties that can be exploited to improve FT techniques:
   -We believe that there is more to come
   -Other application domains may benefit of fundamental application property analysis to improve FT techniques

2) Large systems are generating event notifications that can be used to improve FT techniques:
   -Checkpoint intervals should consider the actual resources used by the execution (like for reduction of energy consumption)
   -We can try to predict failures and take proactive actions (but still need significant improvement)

Joint Laboratory for Petascale Computation

Conclusion

INRIA NCSA

Building a Resilience ecosystem

A holistic approach

Performance Modeling

In presence of failure

Correlation analysis for root cause and prediction

Optimal checkpoint interval

Fault tolerant protocols

Fault model

Sys Admin failure categorization

API for application level checkpointing

Even classifier

Multilevel checkpoint restart

Collection of Event logs (NCSA, TACC, Usenix, etc.)

Fault understanding and proactive fault avoidance

Low overhead reactive fault tolerance

Joint Laboratory for Petascale Computation

Questions?

INRIA NCSA

- Google "joint-lab"
  - Between INRIA and UIUC
  - In the context of Blue Waters

References:

- A. Guermouche, T. Ropars, M. Snir, F. Cappello, HydEE: Failure Containment without Event Logging for Large Scale Send-Deterministic MPI Applications, Proceedings of IEEE IPDPS 2012.
- A. Gainaru, F. Cappello, B. Kramer, Taming of the Shrew: Modeling the Normal and Faulty Behavior of Large-scale HPC Systems, Proceedings of IEEE IPDPS 2012
- E. M. Heien, D. Kondo, A. Gainaru, D. Lapine, B. Kramer, F. Cappello, Modeling and Tolerating Heterogeneous Failures in Large Parallel Systems, IEEE/ACM SC11
- A. Guermouche, T. Ropars, E. Brunet, M. Snir, F. Cappello, Uncoordinated Checkpointing Without Domino Effect for Send-Deterministic Message Passing Applications, IEEE IPDPS 2011
- A. Gainaru, F. Cappello, B. Kramer, Event log mining tool for large scale HPC systems, Europar 2011
- T. Ropars, A. Guermouche, B. Uçar, E. Meneses, Laxmikant V. Kalé, F. Cappello, On the Use of Cluster-Based Partial Message Logging to Improve Fault Tolerance for MPI HPC Applications, Europar 2011
- F. Cappello, A. Guermouche, M. Snir, On Communication Determinism in Parallel HPC Applications, IEEE ICCCN 2010