Assessing Time Coalescence Techniques for the Analysis of Supercomputer Logs

Catello Di Martino
Center for Reliable and High-Performance Computing
University of Illinois at Urbana-Champaign
1308 W. Main Street, Urbana, IL 61801, USA
Email: dimart@illinois.edu

Marcello Cinque, Domenico Cotroneo
Dipartimento di Informatica e Sistemistica
Università degli Studi di Napoli Federico II
Via Claudio 21, 80125 Napoli, Italy
Email:{macinque,cotroneo}@unina.it

***Abstract*—This paper presents a novel approach to assess time coalescence techniques. These techniques are widely used to reconstruct the failure process of a system and to estimate dependability measurements from its event logs. The approach is based on the use of automatically generated logs, accompanied by the exact knowledge of the ground truth on the failure process. The assessment is conducted by comparing the presumed failure process, reconstructed via coalescence, with the ground truth. We focus on supercomputer logs, due to increasing importance of automatic event log analysis for these systems. Experimental results show how the approach allows to compare different time coalescence techniques and to identify their weaknesses with respect to given system settings. In addition, results revealed an interesting correlation between errors caused by the coalescence and errors in the estimation of dependability measurements.**

***Index Terms*—Event Log Analysis, supercomputer dependability, data coalescence, dependability assessment**

I. Introduction

Event logs represent one of the main data sources for analyzing the dependability behavior of computer systems during the operational phase. They are being largely used in the context of supercomputers [1], [2], where the unattended operation of the system forces administrators to look at the logs written by applications and system modules to analyze the occurrence and consequences of system outages [3].

A significant issue in event log analysis is to determine the real occurrences of failures, starting from raw log entries. The problem is that, when a failure manifests in the system, multiple apparently independent error events may be written in the log. Data coalescence techniques aim to reconstruct the failure process of the system by grouping together events related to the same failure. Identified occurrences are then used to classify the failure modes of the system and to evaluate dependability measurements, such as, the mean time between failure (MTBF), the mean time to recover (MTTR). Hence, data coalescence represents a crucial step in failure data analysis, since inaccurate grouping of failures leads to a distorted estimation of the dependability of the system.

In the last decade, several coalescence techniques have been proposed. Many of them are variations of the well known time-based tuple heuristic [4], [5], which has been used in a large variety of studies [1], [3], [5]–[13]. The heuristic is based on the assumption that events related to the same failure are close in time. Hence, if the timestamps of two events fall within a specific coalescence window, they are grouped in the same tuple, representing a single failure.

Despite the large use of these techniques, several studies recognized the problem of the accuracy of log-based dependability analysis [12], [13]. As a matter of fact, a single tuple may contain events related to different failure manifestations (also known as *collisions*), or events related to the same failure may be wrongly coalesced in different tuples (also known as *truncations*). The most common criticism against coalescence studies is the lack of assessment approaches, since the ground truth on the real failure process is usually not available. In other words, we cannot know how well coalescence techniques are able to reproduce the actual failure process. The problem is clearly exacerbated in the case of supercomputers, where the size and the complexity of the system increase the log size by orders of magnitude. These issues contribute to a decreased level of trust on log-based dependability analysis [3].

Past attempts towards the assessment of coalescence techniques moved along two main directions: 1) to assume a theoretic failure process as the ground truth [5], and 2) to extract the ground truth from accurate interviews with system administrators [1], [3], [7], [8], [14]. Along direction 1), only simplistic mathematical models have been adopted so far [5] (such as, exponential failure inter-arrival). Therefore, it is still unclear how system-related aspects, such as the workload and the propagation of failures among components and nodes, impact on the accuracy of results. As for direction 2), the accuracy of the analysis is often biased by the subjective knowledge of system administrators. Hence, the problem of assessing coalescence techniques is still unresolved.

In this paper we propose a novel approach to assess time coalescence techniques. The driving idea is the following: given a system and a failure process, we provide a model-based tool able to properly generate *synthetic logs* along with the ground truth they represent. The assessment is then conducted by comparing the *presumed reality*, reconstructed by coalescing synthetic logs, with the *objective reality* provided by the ground truth. Synthetic logs and related ground truth are generated in a web-based framework based on a set of Stochastic Activity Networks [15]. The framework along with the proposed approach allow to evaluate the sensitivity of coalescence techniques against configuration parameters (e.g., the coalescence window), and to estimate how system related aspects, such as, the number of nodes in the system, the

978-1-4673-1625-5/12/$31.00 ©2012 IEEE

workload, the failure type and propagation patterns impact on the accuracy of measurements.

Experiments focus on the analysis of four different state-of-the-art time based coaelcence techniques and reveal interesting findings. First, we found a strict correlation between the estimation error on the MTBF and the percentage of truncations and collisions; hence, if techniques are defined able to estimate the probability of truncations and collisions, it will be possible to estimate the MTBF error as well. This is an important finding, since it shows that it is possible to relate measurement errors to coalescence errors. Second, we found that even when the techniques are tuned with theoretically optimal values for their parameters, still the statistical properties of failure distributions may not be preserved, due to unavoidable accidental collisions. Third, we found an unacceptable increase of the error as the size and complexity of the system increases; this suggests that time coalescence techniques need to be rethought to be usefully adopted also in future large-scale computer systems.

The rest of the paper is structured as follows. Section II underlines the needed background and relation with the state of the art. Section III outlines our validation approach, described in details in section IV. Section V reports experimental results, discussing the main findings obtained. Finally, Section IV ends the paper with lessons learned.

II. Background and Related Work

Logs are human-readable text files reporting sequences of text entries (the log events), ranging from regular to error events occurred at runtime [5].

The analysis of logs usually accounts three consecutive steps: i) data filtering [1], [3], [8], [16], [17], concerning the removal of log events not related to failures, ii) data coalescence, concerning the grouping of redundant or equivalent failure events, and iii) data analysis, concerning the evaluation of dependability measurements, the modeling of the failure process [10], [18]–[22], and the investigation of failure propagation phenomena among different nodes [1], [13], [21], [23] and among different subsystems within nodes [1], [23].

Data coalescence is crucial in log analysis, since it aims to reconstruct the failure process of the system by grouping together log events related to the same presumed failure. The reason is that, as the effects of a fault propagate through a system, hardware and software detectors are triggered resulting in multiple events reported in the log [5]. Moreover, the same fault may persist or repeat often over time [3].

Coalescence techniques can be distinguished in time, spatial, and content based.

Time coalescence is based on the assumption that log events due to the same cause are close in time. An important methodological achievement for time coalescence has been the definition of the tuple heuristic [4], [5]. According to the heuristic, all the events that fall within a specific time window are grouped in the same tuple. Clearly, the selection of an appropriate value for the time window is crucial. In [5] a heuristic for the selection of a single coalescence time window for the Tandem TNS II system is presented. Authors found that the number of tuples of a given log is a monotonically decreasing function of the time window, with a characteristic

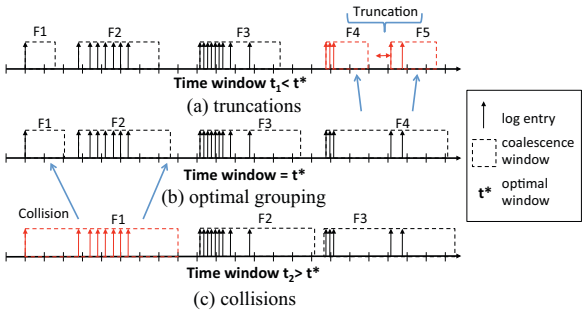

Fig. 1: Example of wrong grouping: (a) truncations and (c) collisions with respect to optimal grouping (b).

"L" shaped curve. According to their findings, the "knee" of the curve represents the internal clustering time of the system, hence, the time window should be chosen right after the knee.

Spatial coalescence is a variation of time coalescence used to relate events that occur close in time but on different nodes of the system under study. It typically consists in applying time coalescence techniques on log files obtained by merging the logs of individual nodes [1], [3], [13], [24].

Time and spatial coalescence techniques have been applied to a variety of large scale computing systems [2], [3], [5]–[8], [11], [16], [20], [25], [26]. The common trend is to use tupling with a fixed value for time window, such as 5 minutes [1], [3], [6]–[8], [27], [28], 20 minutes [9], [10], [12], and 60 minutes [12], [13], usually without any tuning (such as, the knee rule) or validation.

Content-based coalescence techniques are emerging recently [14], [16], [28]–[30], based on the grouping of events by looking at the specific contents of log messages. For instance, [28] and [30] apply the lift data mining operator to find frequent event patterns starting from log contents, hence isolating accidental patterns. In this work we focus on time coalescence techniques, being them the most adopted in the literature due to their simplicity, and we plan to extend our approach to content-based techniques.

It is known that data coalescence can distort the results of the analysis due to imperfect groupings caused by *truncations* and *collisions* [5], [31]. Figure 1 provides an example of truncations and collisions with reference to time and spatial coalescence. A truncation occurs when the time between two or more events caused by a single failure is greater than the clustering time, thus causing the events to be split into multiple tuples (Figure 1.(a)). A collision occurs when two independent failures occur close enough in time such that their events overlap and they are erroneously combined into a single tuple (Figure 1.(c)). Consequently, the goodness of the coalescence process is strictly dependent on the selected time window [5].

Despite these issues, still little attention has been devoted to the assessment and validation of coalescence techniques. One approach often used in the field is to adopt failure reports by system administrators as the ground truth [14], since they contain data which do not require manipulation. However, administrator reports are not always available in all systems, or they can be biased by the subjective knowledge of administrators.

In [5] authors develop a model to relate the collision probability to the event arrival rate in the log, and to study the

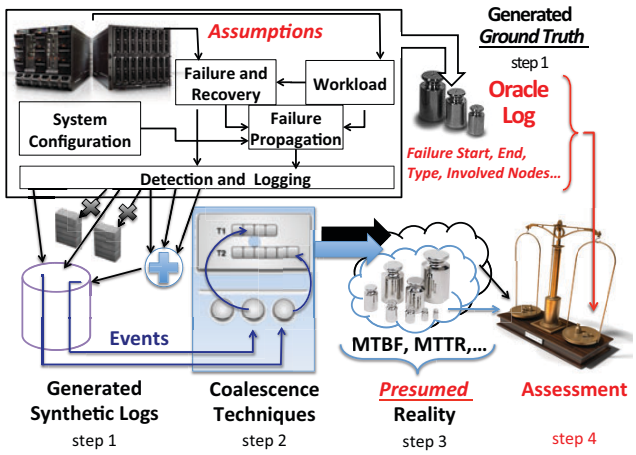

Fig. 2: Representation of Validation Methodology.

sensitivity of tupling to the dimension of the time window. Using a similar model, Buckley et al. in [31] assessed the effectiveness of Tsao's [4] and Hansen's [5] tupling heuristics, both causing as low as 15% of collisions. However, the tuple heuristic and the models used for its validation have been originally defined for past centralized and small-scale systems. As a consequence, the rate of collisions is expected to be sensibly larger in modern supercomputers, due to both the non-exponential fault process [3], [20], and the increased inter-arrival rate of log events.

III. The Log Generator Approach

The proposed approach is based on the generation of synthetic logs for which the ground truth is known. The ground truth represents the actual failure process of the system being analyzed (the *objective reality*), which may not correspond to the *presumed* failure process reconstructed by coalescing produced logs. Hence, a key objective of the proposed approach is to establish how close the presumed reality is to the objective reality, e.g., how much dependability measurements estimated from processed logs, such as MTBF and MTTR, differ from the ground truth. This also means to assess how much wrong grouping (i.e., collisions and truncations) affect measurements.

Figure 2 summarizes the proposed assessment and validation approach. Step 1 concerns the generation of synthetic logs jointly with the ground truth, stored in the *oracle log*. Synthetic logs are similar to real system logs: each entry contains a time stamp, the system component/node that wrote the entry, and the error message. The oracle log, instead, contains detailed information about every single failure affecting the system, including its start time, end time, type, and the set of system resources involved (e.g., nodes and subsystems).

The correct emulation of the failure process of the system is a fundamental step to generate synthetic and oracle logs. In particular, in the case of supercomputers, the failure process depends on the following aspects, to be taken into account:

- the system configuration in terms of number and role of computing nodes, as well as the number and type of *subsystems* composing them (e.g., hardware, I/O, network, etc.); this information is needed since it impacts on the number and type of failures and on their propagation [20], [22], [27];
- the workload, in terms of number and type of jobs and their inter-arrival, queuing, scheduling, duration, and number of required nodes; as known, the workload influence the failure behavior of the nodes in terms of failure inter-arrival and failure duration distributions [32], due to the different subsystems stressed during the computation;
- the characterization of failure and recovery processes of nodes, which impact on the number and distribution of errors stored in the logs;
- the characterization of failure propagation phenomena; it is known that failures can propagate between the subsystems composing a node [33], [34] or among the nodes composing the system [1], [8];
- the detection and logging mechanisms which emulate the writing process of events in the synthetic logs, due to failures.

Once synthetic logs have been generated, they are processed using different coalescence techniques, to reconstruct the *presumed reality* (step 2 in Figure 2), which is analyzed to evaluate dependability measurements (step 3). Finally, in step 4, the presumed reality is compared to the oracle log to assess measurement errors.

IV. The Log Generator Framework

The proposed approach has been implemented in a framework composed of three main elements: a user interface, a *log generator* component, and a tool, named *analyzer*, for automating the analysis of the results. The user interface is implemented as a web-based application[1], and it is in charge of collecting information about the configuration of the system and all the parameters needed by the *log generator* component to produce the logs (the exhaustive list of parameters is presented later in section V).

A. The log generator

The *log generator* component is the core of the framework and it is in charge of generating both synthetic and oracle logs. It is based on a set of hierarchical models, which take into account system-related aspects, i.e., the number of nodes, subsystems inside a node, the workload, and the related failure activation and propagation processes. Models are implemented as Stochastic Activity Networks (SAN) in the Mobius Tool [15]. The whole system is modeled as a replicate/join composition [15] of a set of distinct nodes, modeling computing resources. Each node is in turn composed of a set of *subsystem models*, i.e., memory, processor, IO, software, and network, each one specifying the failure and recovery process of the modeled subsystem, when subject to a specific workload.

The *workload model* is introduced to take into account the dependency between the workload (its type and intensity) and the failure rate of subsystems [20], [24], [27], [32]. For instance, nodes executing IO bound jobs are known to be more susceptible to IO failures than nodes processing CPU intensive jobs [1], [23], whereas, wide jobs (i.e., jobs requesting a large

[1]The framework is accessible at the following address http://www.catellodimartino.it

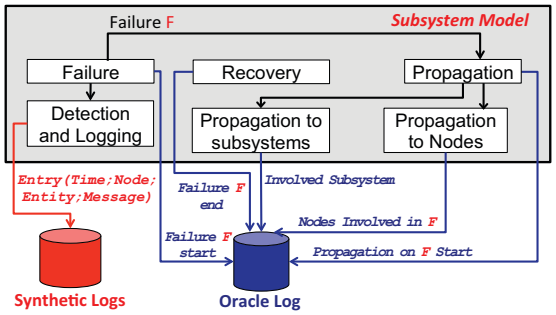

Fig. 3: Structure of a generic subsystem model.

number of nodes) are known to be more prone to failures than small jobs [32].

Figure 3 shows the structure of a generic subsystem model. It is composed of i) a failure model, ii) a propagation model (to model failure propagations to other subsystems and to other nodes), iii) a detection and logging model, and iv) a recovery model. The failure model reproduces the failure inter-arrival process of both local failures and external failures, i.e., propagated from other subsystems (local propagation) or from other nodes (spatial propagation). For instance, concerning local propagation, it is known that software failures may depend on memory or on processor failures [33], [34], but they can also manifest due to endogenous errors, such as software bugs. They can still propagate to other subsystems, such as to the network or IO. Instead, concerning spatial propagation, it is known that network and IO failure manifestations are clustered in space [1], [8], [13], [21]; this means that a single IO or Network failure manifesting on a node can easily propagate to multiple nodes. The detection and logging is in charge of producing entries in the synthetic log when failures occur. The writing of entries is terminated when the subsystem recovers. All the models concur to produce the ground truth on the failure process, stored in the oracle log.

In the remainder of this section, we describe the IO model, as an example of subsystem model, and the workload model in details. Due to space limitations, only important details of the SAN models are provided.

1) The IO Subsystem Model: The IO subsystem model is organized in two layers. The first is the SAN layer, shown in Figure 4, in charge of reproducing the stochastic behavior of the failure, recovery, and log writing processes. The second is implemented by means of an external C++ library, called `ExternalLib`, in charge of decoupling the stochastic behavior from system related aspects, such as, job scheduling and spatial propagation of failures. For the sake of clarity, Figure 4 divides the model in five zones. Zone 1 models the activation of local IO failures. Zone 2 models the failure activation due to external causes, such as propagation from other subsystems or from other nodes. Zone 3 is in charge of modeling the propagation of local failures (due to zones 1 or 2) to other subsystems or nodes. Zone 4 mimics the detection and logging mechanisms. Finally, zone 5 models the failure recovery process. The models of the other subsystems share the same internal organization.

a) Zone 1 - Failure Activation: The IO subsystem may fail following three different distributions, modeled by *TTF1*, *TTF2*, and *TTF3* actions. These distributions model the time to failure for nodes executing i) IO intensive jobs, ii) mixed IO and CPU bound jobs, and iii) CPU bound jobs, allowing to generate failures depending on the running workload. The correspondence between the node with ID (unique) *nodeID* hosted in the rack *rackID* and running jobs is handled by the `ExternalLib`. When one of the actions *TTF1*, *TTF2*, and *TTF3* fires, the IO subsystem is marked as failed (mark in the place *failed*) and a unique identifier is generated for the failure (place *failureSignature*). The signature is used by the `ExternalLib` to keep track of failure propagations.

b) Zone 2 and Zone 3 - Failure Propagation: When a failure is activated in zone 1, it can cause a local propagation. To this aim, the places *propagate_to_CPU*, *propagate_to_SW*, and *propagate_to_MEM* of zone 3 are shared with the places *CPU_correlated*, *MEM_correlated*, *SW_correlated* of zone

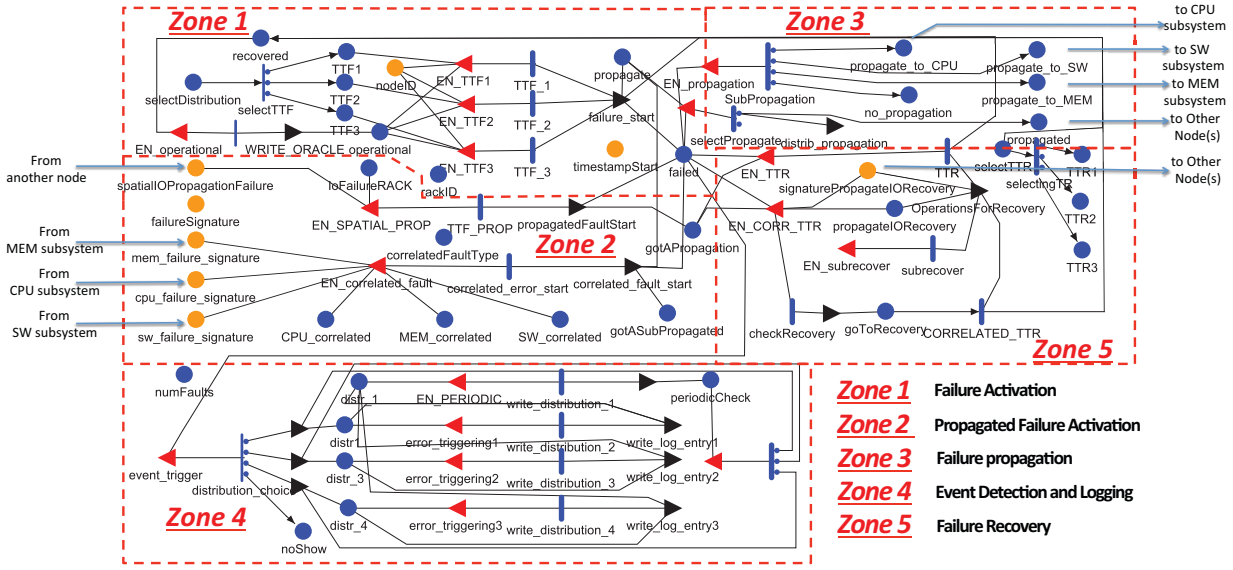

Fig. 4: The SAN model of the IO subsystem

TABLE I: Example of code from the Output Gate distrib_propagation

	Output Gate **distrib_propagation**
1	ExternalLib* l = ExternalLib::Instance();
2	Distribution distr = l− >getAffectedNodesDistr();
3	int affected_nodes = (int)distr.next();
4	std::vector<int>IDs;
5	IDs=(l->getPropIDs(nodeID->Mark(),_IO, affected_nodes));
6	IDs::iterator node;
7	//It cycles on the nodes involved in the propagation
8	...
9	if(*node!=nodeID->Mark())
10	//sets the *node as failed in the library
11	l− >setIoFailure(*node, failureSignature− >Mark());
12	IoFailureRACK->Mark()=rackID->Mark();
13	//this wakes up all not failed nodes of rackID
14	//they check if they are failed calling the method
15	//l->isIOFailed(nodeID->Mark()) of the library

2 of other subsystem models. For instance, when a token is placed in the *propagate_to_SW* place, it will be placed also in the *IO_correlated* place of the software subsystem model, enabling a propagated failure activation in zone 2. The propagated failure will share the same signature of the original failure, and it will cause the affected subsystem to pass in the failed state.

Spatial propagation follows a similar mechanism: upon a failure activation, the failure is propagated with a given `prop_probability` (action *selectPropagate* in zone 3) to a set of nodes. Consequently, a failure is activated in the zone 2 of all nodes affected by the propagation. The selection of affected nodes is performed by considering i) the nodes running the same job as the failed one [32] (to take into account the workload), and ii) nodes sharing the same rack. Since these aspects are dependent on the system configuration and on the running workload, we delegate the selection of affected nodes to the `ExternalLib`. In this way, it is possible to account for different failure propagation criteria without changing the SAN model. The use of the library is exemplified in Table I, reporting the code of the output gate *distrib_propagation* in zone 3. First, a number of `affected_nodes` is extracted (line 3) using a distribution (returned by the library - line 2). Then, the IDs of affected nodes are selected using the `getPropIDs()` method of the `ExternalLib` (line 5). Finally, selected nodes are tagged as failed using the `setIoFailure()` method of the library (line 11); this method requires the failure signature to allow the `ExternalLib` to keep track of all spatial propagations. A node receiving a propagation (zone 2) is notified by means of the *EN_SPATIAL_PROP* input gate. Specifically, the input gate is enabled if the node has been tagged as failed (the check is performed in the input gate using the `isIOFailed()` method of the `ExternalLib`) and if the mark of *IoFailureRACK* is equal to *rackID*. The propagated failure will manifest after a specific delay modeled in *ttf_prop*.

c) Zone 4 - Detection and Logging: When a subsystem fails, one or several events can be written into the synthetic log. The *distribution_choice* activity selects a random number of entries to be generated in the log, according to different writing distributions. For instance, it may select a periodic writing process (*distr1*, to emulate timeouts and watch dog timers behaviors, typical in logs), or it may select a random inter-arrival of events (*distr2*, *distr3*, or *distr4* branches). When the subsystem recovers from the failure (zone 5), the writing of events in the synthetic log is interrupted.

TABLE II: Example of synthetic logs

Time	NodeID	Subsys	Message
11/22/11 19:06:41	191	IO	IO Error no. 1
11/22/11 19:06:41	212	IO	IO Error no. 1
11/22/11 19:06:41	212	IO	IO Error no. 2
11/22/11 19:06:41	191	IO	IO Error no. 2
11/22/11 19:06:41	195	IO	IO Error no. 1
11/22/11 19:06:41	195	IO	IO Error no. 2
11/22/11 19:06:41	212	IO	IO Error no. 3
11/22/11 19:06:43	195	SW	SW Error no. 1
11/22/11 19:06:43	192	IO	IO Error no. 1
11/22/11 19:06:46	192	IO	IO Error no. 2
11/22/11 19:06:46	191	IO	IO Error no. 3
11/22/11 19:06:49	195	SW	SW Error no. 2
11/22/11 19:07:01	195	SW	SW Error no. 3
11/22/11 19:07:02	161	NET	NET Error no. 1
11/22/11 19:07:03	195	IO	IO Error no. 3
11/22/11 19:07:09	195	IO	IO Error no. 4
11/22/11 19:07:09	161	NET	NET Error no. 2
11/22/11 19:07:09	195	SW	SW Error no. 4
11/22/11 19:07:47	161	NET	NET Error no. 3
...			
12/17/11 23:22:39	297	PROC	CPU Error no. 1
12/17/11 23:22:42	297	MEM	MEM Error no. 1
12/17/11 23:22:43	297	MEM	MEM Error no. 2
12/17/11 23:22:49	297	MEM	MEM Error no. 3

As it happens in actual systems, such as in the Unix syslog, entries in the generated synthetic logs encompass a time stamp, the ID of the node generating the event, the subsystem in which the event took place, and a text message. Table II shows an extract of the generated log. The message contains a generic text and the sequence number of the entry, since time coalescence techniques do not exploit message contents. It is worth noting that we generate logs containing only error entries without any other content, i.e., we assume logs are pre-processed with a perfect filtering and free from all useless entries. Despite optimistic, this assumption avoids to correlate measurements with the adopted filtering, allowing us to focus only on the coalescence.

d) Zone 5 - Recovery: Failures generated in zone 1 recover following a random recovery process, implemented by the *TTR* activity in zone 5. Failures propagated by other subsystems (place *gotASubPropagated*) may be recovered only if the subsystem, which generated the failure, recovers itself (action *subRecover*). Failure propagated by other nodes (place *gotAPropagation*), may be recovered with a probability r following the *TTR* distribution, or with a probability $1 - r$ only after the recovery of the node which propagated the failure. The distribution of the TTR will be selected depending on the TTF action that caused the failure. In both cases, the place *signaturePropagateIORecovery* is used by the node/subsystem that initiated the failure to communicate to involved nodes/subsystems that it recovered. To this aim, this place will contain the *failureSignature* of the node/subsystem, which started the failure, and nodes/subsystems failed due to the same *failureSignature* will consequently recover after a time *correlatedTTR*. In all the mentioned cases, after a recovery, the output gate *WRITE_ORACLE* in zone 1 will be executed and an entry for the oracle log will be prepared. When all the subsystems and nodes involved in a failure recover (kept track by the `ExternalLib` via the signature), an entry in the oracle is finally written. It will contain the start time of the failure, the originating node, the type of failure (i.e., with or without propagation, with the indication of affected nodes and subsystems), the number of error events generated

TABLE III: Example of Oracle log

Failure Start Time	NodeID	Type of Failure	Events	Duration	Signature
11/22/11 19:06:39	191	IO_PROPAGATION(192,212,195)+SW(195)	112	124.332	75829788
11/22/11 19:07:02	161	NET	9	154.8	35924727
12/17/11 23:22:39	297	PROC+MEMORY	20	143.85	71162829

TABLE IV: Example of code from the Output Gate `schedule_long`

Output Gate **schedule_long**

```
1  ExternalLib* l = ExternalLib::Instance();
2  jobs* nextJob = l->do_schedule(_LONG);
3  LongQueue::iterator node;
4  // It cycles on the Free Nodes of the Long Queue
5   ...
6   if(!(ExternalLib::isBusy(node))) {
7   l->setBusy(myjob->getJobID(),node);
8   myjob->addNode(node); }
9   ...
10 l->enqueueRunning(myjob);
```

(and written in the synthetic log), the failure duration, and the failure signature. Table III shows an extract from the oracle log corresponding to the synthetic log shown in Table II. From the oracle, it can be seen that IO error entries for nodes 192, 212, and 195 are due to the same failure generated by node 191. In addition, the software error written for node 195 is the result of a propagation of the same failure of the IO subsystem. Conversely, the failure of node 161 is independent from the previous one, despite it overlaps in time.

2) The Workload Model: Figure 5 shows the SAN of the workload model. It mimics the inter-arrival of jobs and the behavior of a job scheduler, which assign jobs to nodes according to FIFO with backfill scheduling queues[2]. In particular, three different queues are considered, depending on the type of jobs: i) Long queue, for long lasting cpu intensive jobs, ii) the IO queue, for IO bound jobs, and iii) the Default Queue, for mixed jobs (cpu and IO). Recall that nodes running different job types may experience different failures. This way, we make it possible to specialize at runtime the type of workload run by a node, and hence its failure behavior.

The model is composed by 3 main zones, as depicted in the Figure. The first models the job inter-arrival, the second the job scheduling, and the third the job completion. Also this model takes advantage of the external library, which implements the scheduling policy.

[2]www.adaptivecomputing.com/resources/docs/mwm/6-0/8.2backfill.php

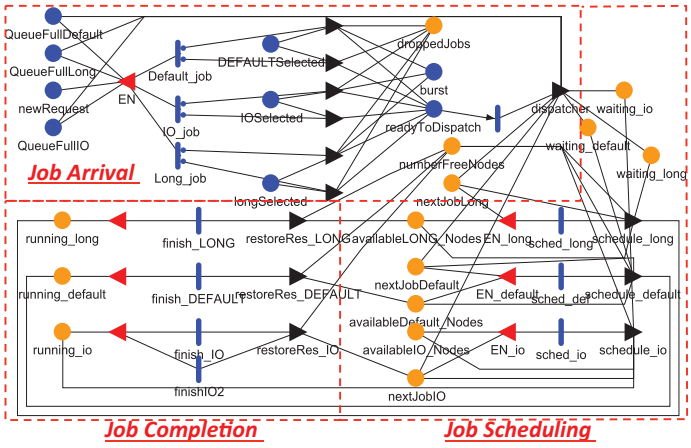

Fig. 5: The SAN of the workload model.

The job arrival process has been modeled by means of three actions *Default_job*, *IO_job* and *Long_job* enabled by the input gate *EN* (Figure 5), only if the correspondent queue is not full. Once a job is arrived in the system, a `JOB` object is created by the external library, containing fields such as `JobID` identifying the job univocally the arrived job, `Type` coding the scheduling queue of destination, `duration` containing the duration of the job, `WC` specifying the job wall clock time, `numtask` specifying the number of required processors, and a vector `runNodes`, initially empty, filled during the scheduling with the ID of the node(s) in charge of executing the job. The dispatching of the arrived jobs to the specific scheduling queue is performed by the output gate *dispatcher*, which is in charge of pushing the created JOB object in the waiting queues managed by the external library.

The scheduling is executed by the output gate in the job scheduling zone. Table IV shows an extract of the code for scheduling a waiting job in the long queue. It is worth noting that the scheduling is achieved by means of the `do_schedule()` method of the library, hence allowing to simulate different scheduling policies without changing the model (for instance, overriding the `do_schedule()` function).The function `setBusy()` is in charge of loading the job to the selected node. The function `enqueueRunning()` of the library is called to finally run the job on the selected nodes. As for the jobs completion, the library keeps track of the duration of rub jobs in a list ordered by completing times, and modify the firing times of the actions `finish_LONG`, `finish_DEFAULT` and `finish_IO` consequently.

B. The Analyzer

The analyzer is in charge of coalescing the generated synthetic logs with one or more coalescence techniques and of comparing the obtained results against the oracle log. To this aim, we implement a set of Perl scripts in charge of: i) processing the oracle log to extract the ground truth (number of generated failures, real MTBF, real MTTR, etc.), ii) running the target coalescence techniques on synthetic logs to extract the presumed reality (number of tuples, estimated MTBF and MTTR, etc.), iii) comparing the presumed reality with the ground truth to evaluate estimation errors, and the number of truncations and collisions, and iv) conducting detailed analysis on coalescence techniques to estimate their sensitivity to changing parameters, such as the time window.

The analyzer has been implemented with the objective to automate the analysis on a set of generated logs, each one accounting for different system characteristics (number of nodes, type of workload, type and intensity of failures, etc.). In addition, we have adopted a modular design to simplify the introduction of novel coalescence techniques to be assessed.

V. Experimental Evaluation

In this section, we report the results of the assessment of four time coalescence techniques by means of the proposed

TABLE V: Event Logs Coalescence Algorithms. T(x) is the time stamp of event x; Type(x) is the type of event x; "next" and "curr" represent the next and current events in the log, respectively; "first" represent the first event in the current tuple.

ALGORITHM 1. ***Sliding Window***

```
sliding() {
1.   W = window size;
2.   foreach event in the log {
3.     if (T(next)-T(curr) < W)
4.       add next to the current tuple;
5.     else
6.       create a new tuple; }
7. }
```

ALGORITHM 2. ***Fixed Window***

```
fixed() {
1.   W = window size;
2.   foreach event in the log {
3.     if (T(next)-T(first) < W)
4.       add next to the current tuple;
5.     else
6.       create a new tuple;}
7. }
```

ALGORITHM 3. ***Tsao's Rule (tsao)***

```
tsao() {
1.  foreach event in the log {
2.    if (T(next)-T(curr) < 2.8 min.)
3.      add next to the current tuple;
4.    else
        if (T(next)-T(curr) < 22.5 min.
      and Type(next) already in the tuple)
5.        add next to the current tuple;
6.    else
7.        create a new tuple;}
8. }
```

ALGORITHM 4. ***Spherical Covariance (SC07)***

```
SC07() {
1.  ttr = current est. of MTTR;
2.  theta = current est. of inter-failure time;
3.  C = threshold parameter;
4.    foreach event in the log {
5.      D = T(next) – T(curr);
6.      if (D <ttr and Type(next) = Type(curr))
7.        add next to the current tuple;
8.      elseif (D < theta)
9.        c = 1 – αD/theta + β(D/theta)³;
10.       if (c > C)
11.         add next to the current tuple;
12.       else create a new tuple;
13.     else create a new tuple;}
14.}
```

framework. In the following, we consider two case studies: 1) logs generated according to simplistic failure assumptions, and 2) logs generated according to system-representative assumptions, to evaluate how system-representative aspects impact on results.

A. Coalescence Techniques

Table V reports the algorithms used to implement the coalescence techniques analyzed in this section.

Algorithm 1 (*sliding*) is the classical tupling algorithm with a single, sliding coalescence window [5]. The grouping rule states that if two consecutive events fall within the same temporal window W (line 3), then they are grouped in the same tuple (line 4). Algorithm 2 (*fixed*) is a variant of Algorithm 1, where the coalescence window is fixed: an event is added to the current tuple if its time stamp is far at most W seconds from the first event in the tuple. Algorithm 3 (*tsao*) adds the so-called Tsao's rule [4] to the grouping scheme. In this case two windows are used, the first fixed at 2.8 minutes, the second at 22.5 minutes. Similarly to the classical sliding scheme, if two events fall within the first window, then they are grouped. Else, if they are distant in time (e.g., within the second window), but they are of the same type, then presumably they are related to the same problem, hence they are grouped in the same tuple. This rule has been defined to reduce truncations, since it gives a second chance to events, which normally would be grouped in two different tuples. Finally, algorithm 4 (*SC07*) refers to the scheme recently defined in [23] for the clustering of events in coalitions of clusters, based on a spherical covariance estimate. According to this scheme, two events of the same type are grouped if they fall within the typical time to recover (line 6). This allows the algorithm to group separately the events of different type (e.g., caused by different subsystems). Otherwise, if two events fall within the typical failure inter-arrival (the $theta$ parameter - line 8) they are grouped only if their spherical covariance value, defined as: $c = 1 - \alpha \cdot (D/theta) + \beta \cdot (D/theta)^3$ is above a given threshold C (lines 9 and 10), where D is the temporal distance between the two events and α and β are two parameters such that: $\alpha = 1 + \beta$. This algorithm should reduce both collisions and truncations, since events within the MTTR are grouped only if they are of the same type, whereas events within the MTBF are grouped only if they have a certain degree of correlation. The problem is that the algorithm requires an initial estimation of the mean time to recover and of the failure inter-arrival from administrative logs, which could be not available or not accurate.

In the following case studies, we use the proposed approach to validate the described algorithms under different assumptions for their parameters (except from *tsao*, which uses fixed values for its coalescence windows):

- *optimal*: the parameters are set in order to obtain the minimum error on the MTBF and the lowest number of truncations and collisions. Such an ideal tuning allows to evaluate the minimum error achievable for a given technique in a given setting;
- *knee* (for *sliding* and *fixed* only): the window is set using the knee heuristic described in Section II;
- *literature* (for *sliding* only): the window is set to the most adopted values in the literature, i.e., 5 and 20 minutes.

In some cases, we perform sensitivity analysis to varying values of parameters, to achieve better insight on the behavior of the algorithms. For instance, in the case of *SC07*, it is important to estimate how much a wrong estimate of MTTR and MTBF, used to set the *ttr* and *theta* parameters, impacts on measurement errors. The same is done for *sliding* when varying the size of the window.

B. Metrics

According to the defined approach, for each experiment run we generate a synthetic log and an oracle log under given experimental settings. The oracle log is used to calculate the following metrics characterizing the ground truth:

- F: the real number of failures;
- $MTBF_{real}$: the real mean time between failure;
- $MTTR_{real}$: the real mean time to recover.

The synthetic log is processed with all the described algorithms, and the results are compared with the oracle log to evaluate the following metrics:

- $MTBF_{err}$: the error on the MTBF estimate, defined as $MTBF_{err} = (MTBF_{est} - MTBF_{real})/MTBF_{real}$, where $MTBF_{est}$ is the MTBF estimated from the coalesced log ad the average distance between tuples;
- $MTTR_{err}$: the error on the MTTR estimate, defined as $MTTR_{err} = (MTTR_{est} - MTTR_{real})/MTTR_{real}$, where $MTTR_{est}$ is the MTTR estimated from the coalesced log as the average length of tuples;
- $trunc_{\%}$: the percentage of truncations with respect to F, defined as $trunc_{\%} = 100 \cdot number\_of\_truncations/F$;
- $coll_{\%}$: the percentage of truncations with respect to F, defined as $coll_{\%} = 100 \cdot number\_of\_collisions/F$.

C. Experimental Settings

Tables VI and VII specify the statistical distributions used to configure the workload and the subsystems models.

TABLE VI: Parameters for the workload model

Parameter	Value
Light workload inter-arrival, long queue	Weibull(0.881,1212)
Light workload inter-arrival, default queue	Weibull(0.931, 9408)
Light workload inter-arrival, IO queue	Exp(1.81E-3)
Stressful workload inter-arrival, long queue	Lognorm(4.84;1.584)
Stressful workload inter-arrival, default queue	Lognorm(6.584;2.302)
Stressful workload inter-arrival, IO queue	Lognorm(6.178;1.063)
Job duration, long queue	Lognorm(10.381;0.811)
Job duration, default queue	Lognorm(6.7615;2.285)
Job duration, IO queue	Lognorm(7.525;1.702)
Job duration, IO queue	Lognorm(12.186;0.346)
Tasks in a job, long queue	Lognorm(1.633;0.581)
Tasks in a job, default queue	Lognorm(1.335;0.693)
Tasks in a job, IO queue	Lognorm(1.014;0.498)
Probability $>$ 1 task, long queue	0.602
Probability $>$ 1 task, default queue	0.272
Probability $>$ 1 task, IO queue	0.533

Workload distributions have been set with reference to the load of the SCOPE supercomputer [35][3], manufactured by Dell in 2008 at University of Naples Federico II. It is constituted of 512 Dell Blade servers equipped with 2 quad core Intel Xeon CPUs (4096 cores in total), 8 or 16 GB of memory per blade and Infiniband interconnections. SCOPE runs Maui/PBS System for the scheduling of 19 different FIFO with backfill queues. Queues are configured in three classes, differing for the facilities allowed to use and for the max allowed wall clock times.

The workload characterization has been performed by analyzing 6 months of the scheduler logs (Moab/PBS), containing exact information on about 786685 jobs of interest [36]. Using a $k-$means clustering on the job inter-arrival rate per hour, we classified the workload in two classes: light and stressful. We detailed the analysis of job inter-arrivals for all the scheduling queues classes and for both the workload classes, achieving 6 distributions for the job inter-arrival, reported in Table VI. We noticed that the duration of jobs is not correlated to the job inter-arrival classes (correlation ≈ 0.003). Consequently we opted to model job duration independently from the classes of workload, achieving 3 distributions, one for each scheduling queue, reported in Table VI. A similar analysis has been performed to characterize the number of tasks composing a job for each queue, being each task run on a different node.

Considering the parameters for subsystem models, we follow different assumptions for the two case studies. Concerning case study 1, we produced logs for systems ranging from 512 up to 32768 nodes. Only exponential failure inter-arrivals are assumed, by activating only *processor* and *memory* subsystem models. In addition, we do not enable propagations, and we consider only a light workload with CPU bound jobs.

Concerning case study 2, we considered systems ranging from 1024 up to 32768 nodes. We opted for a more realistic configuration, in order to discuss the findings with respect to actual system settings. Failure inter-arrival processes have been configured by means of Exponential, Lognormal and Weibull distributions. Values for their parameters, reported in Table VII, have been selected according to results published for similar systems [10], [18]–[22] as well as by empirical analysis performed on the administrator failure reports of SCOPE. It is obvious that a practitioner aiming only at validating coalescence algorithms can choose different distributions.

In addition, we consider propagations between subsystems and nodes, and we consider both the light and stressful workload, activated on all the three modeled queues.

[3]www.scope.unina.it

TABLE VII: Parameters for the subsystems models

Model	Parameter	Case study 1	Case study 2
IO	TTF1	**N/A**	Weibull(1.091;24220743)
	TTF2		Weibull(0.469;225605)
	TTF3		Weibull(0.71;21398475)
	TTR1		Weibull(1.251;59.52)
	TTR2		Weibull(0.783;853)
	TTR3		Weibull(1.071;5.232)
	prop_probability		0.332
	affected_nodes		Lognorm(1.572;1.071)
Software	TTF1	**N/A**	Weibull(0.759;299865)
	TTF2		Lognorm(10.230;0.667)
	TTR1		Lognorm(5.834;0.82)
	TTR2		Weibull(0.52;112)
Memory	TTF	Exp($4.48E-9$)	
	TTR	Exp(0.016)	
Processor	TTF	Exp($4.48E-9$)	
	TTR	Weibull(0.891;1.304)	
Network	TTF	**N/A**	Weibull(0.46;9215691)
	TTR		Lognorm(5.423;1.101)
	prop_probability		0.092
	affected_nodes_1		Uniform(2;nodesInRack)
	affected_nodes_2		Lognorm(1.911;0.5223)
	affected_nodes_3		Lognorm(5.029.0.45)
All	Simulated_Time	6 months	
	write_distribution 1	Weibull(0.501;20.861)	
	write_distribution 2	Gamma(2.2;109.2)	
	write_distribution 3	Normal(60;1)	
	write_distribution 4	Exp(1.427E-1)	
	write_distribution 5	**N/A**	Gamma(0.105;6.621)
	write_distribution 6		Weibull(1.92;5.01)
System	Nodes per Rack	128	
	Nodes	512-32768	1024-32768
	IO Queue Nodes	**N/A**	[10%Nodes]
	Default Queue Nodes		[30%Nodes]
	Long Queue Nodes	[100%Nodes]	[60%Nodes]
	Workload Type	Light	Light, Stressful
	Scheduling Policy	FIFO with Backfill	

TABLE VIII: Synthetic results for the case study 1 and 2.

Case Study	Nodes	MTBF[s]	Number of Failures					
			IO	SW	NET	PROC	MEM	Total
1	512	122333	**N/A**	**N/A**	**N/A**	59	34	93
	1024	60225				156	101	257
	2048	29683				303	221	524
	4096	16820				491	434	925
	8192	8057				1181	748	1929
	16384	4005				2075	1307	3882
	32768	2057				4474	3080	7554
2	1024	17985	643	442	398	144	95	1722
	2048	11602	1008	1099	698	262	200	3267
	4096	7229	1550	1325	841	440	311	4467
	8192	4621	2165	1430	958	1189	709	6451
	16384	2967	3347	2960	1023	1921	1274	10525
	32768	988	9847	7655	1449	4032	2716	25699

In all the case studies, we parametrize the workload distributions evaluated for SCOPE to the size of the system to analyze, using a maximum likelihood estimation [20].

Table VIII reports synthetic statistics on the MTBF and the number of failures, split by subsystems, generated for the two case studies.

D. Case Study 1: Simplistic Assumptions

Figure 6 shows the results obtained for the first case study. Plots report the error caused by the considered coalescence techniques on the MTBF (Figure 6.(a)), on the MTTR (Figure 6.(d)), the percentage of truncations (Figure 6.(b)) and collisions (Figure 6.(e)), and a sensitivity analysis conducted for SC07 (Figures 6.(c) and 6.(f)).

From the plots we can observe that both *tsao* and *sliding* with $W = 20min.$ overestimate the MTBF when increasing the size of the system, since they tend to produce more collisions (Figure 6.(e)) and no truncations (Figure 6.(b)); the absence of truncations is due to the large size of the window

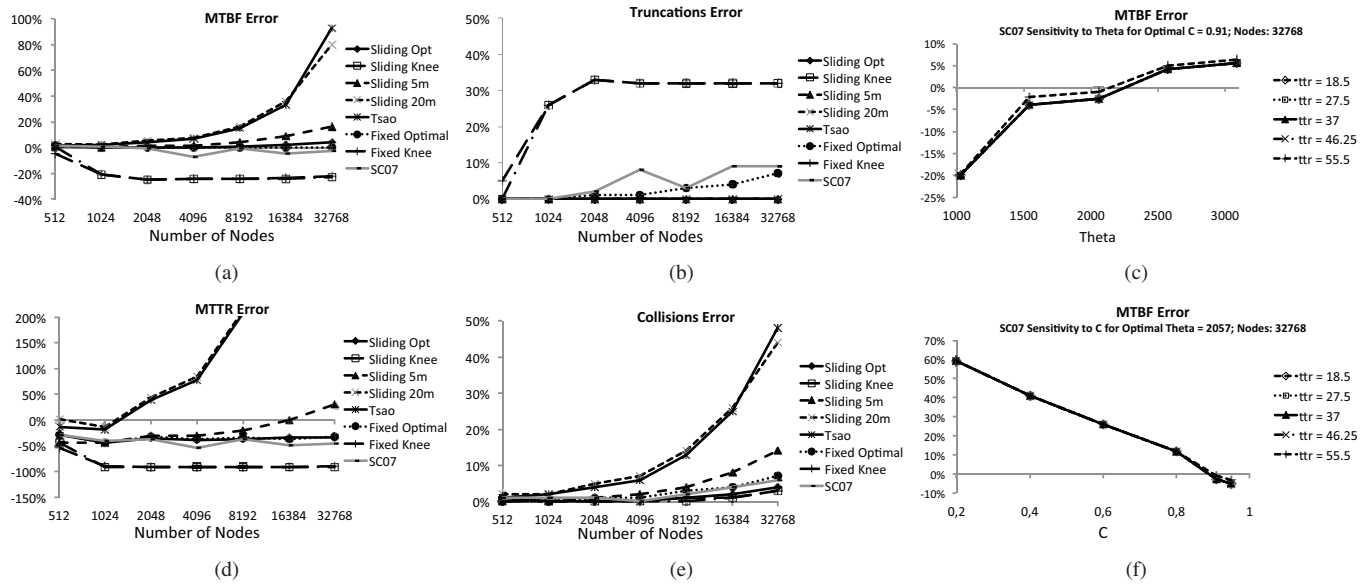

Fig. 6: Results from Case Study 1. (a) $MTBF_{err}$, (b) $trunc_{\%}$, (c) Sensitivity analysis of SC07 to Theta, (d) $MTTR_{err}$, (e) $coll_{\%}$, (f) Sensitivity analysis of SC07 to C

(of the second window for *tsao*), coherently with earlier results in [5]. The overestimation is a dangerous mistake, since it causes the system to be presumed as more reliable than it really is, underestimating the risk of potential failures. The same behavior can be observed for the MTTR: collisions produce longer tuples, and hence longer times to recover.

From the plots we can also observe that *the use of typical time windows, such as 5 minutes and 20 minutes, induces non negligible errors in the estimates*.

For optimal algorithms, that is *sliding opt*, *fixed opt*, and *SC07*, the error is lower, as expected, even if we can notice that errors increase with the system size. This is due to the larger number of events: *even when an ideal optimal tuning of parameters is performed, the large number of events may cause unavoidable accidental overlaps in the logs, compromising the results*. *SC07* exhibits good performance, which are not sensibly affected by the size of the system. However, its ttr, $theta$ and C parameters have been set to obtain the minimum error. Our approach can thus be used to assess how much wrong estimation and/or tuning of these parameters affect the results. Specifically, we can note that a -50% error on $theta$ may induce a -20% error on the MTBF, as reported in Figure 6.(c). Similarly, a wrong tuning on C may strongly distort the results (Figure 6.(f)). The algorithm appears however to be robust against estimation errors on the ttr parameter.

As for the algorithms tuned with the knee rule (that is, *sliding knee* and *fixed knee*), they tend to underestimate the MTBF and MTTR, since they cause a larger number of truncations, and hence more tuples with short duration, if compared to other solutions (Figure 6.(b)). The reason is that, in this case, the knee rule caused the selection of a small coalescence window, as clarified later.

In Figure 7 we report the results of a sensitivity analysis conducted for the *sliding* algorithm with respect to the time window W. The upper part of the figure reports the tuple count (the "L" shaped curve used by the knee heuristic to select the time window), whereas the lower part reports $MTBF_{err}$, $MTTR_{err}$, $coll_{\%}$ and $trunc_{\%}$ as a function of W. The Figure allows to achieve interesting insights on the relationship between estimation errors and the number of collisions and truncations when varying W.

First, it can be noted that $MTBF_{err}$ *tends to 0 for a value* W^* *such that the sum of* $coll_{\%}$ *and* $trunc_{\%}$ *is minimum* (that is, the point where the $coll_{\%}$ curve crosses the $trunc_{\%}$ curve). Interestingly, this point corresponds to the second knee of the tuple count curve. This can be explained by observing that in this case study we configured the memory and processor subsystems to generate failures with the same inter-arrival but with different recovery times (smaller for processor, longer

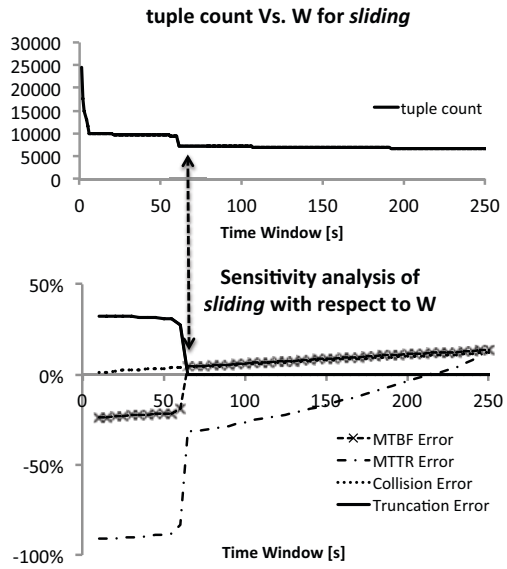

Fig. 7: Sensitivity analysis for the *sliding* algorithm in case study 1, nodes: 32768

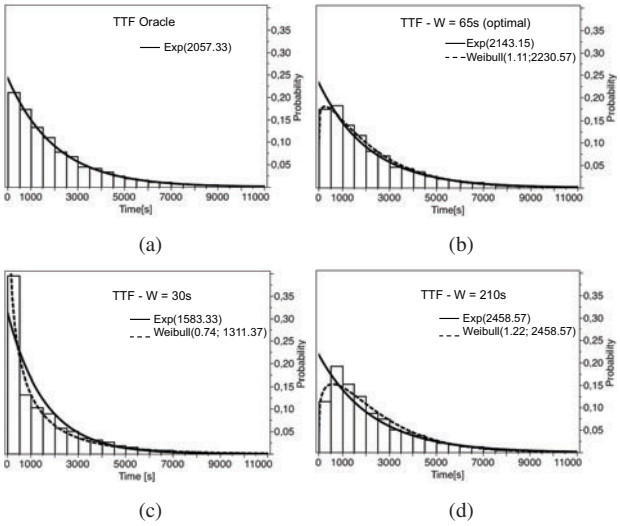

Fig. 8: TTF Distribution from Case Study 1 for the Sliding Window. (a) Oracle, (b) $W = 65s$ (Optimal), (c) $W < 65s$, (d)$W > 65s$.

for memory). The two dynamics are reflected in the two knees. Hence, while the knee rule suggests to choice W after the first knee ($W = 10s$.), the optimal choice is to select the window size corresponding to the second knee ($W = 65s$.). This suggests us that *the selection of the time window should be done considering not only failure inter-arrivals, but also recovery times*. However, this is not easily applicable in practice, as better shown in case study 2.

Second, we can observe that the $MTBF_{err}$ trend is interestingly correlated with the collisions and truncations trends. In particular, if W^* is the optimal window, we can note that $MTBF_{err} = -\alpha \cdot trunc_{\%}$ for $W < W^*$, and $MTBF_{err} = \beta \cdot coll_{\%}$ for $W > W^*$, being α and β two positive real numbers. This means that, ideally, *it will be possible to estimate the error on the MTBF if techniques are defined able to empirically estimate the probability of truncation and collision*. Also, *the estimation of $trunc_{\%}$ and $coll_{\%}$ allows to optimally tune W to the point such that $trunc_{\%} = coll_{\%}$, i.e., the point where $MTBF_{err} = 0$.*

Third, we can note that, *even when the optimal W^* is selected and $MTBF_{err}$ = 0, still the statistical properties of the time to failure (TTF) distribution are not preserved*, due to the presence of accidental collisions. This effect is shown in Figure 8: we can observe that the TTF of the oracle (Figure 8.(a)) follows an exponential distribution, coherently to the simulated failure process. However, the TTF estimated from logs coalesced with the optimal window are distributed as a Weibull with an increasing hazard rate (Figure 8.(b), shape parameter $= 1.11$), as well as for $W > W^*$, (Figure 8.(d)) hence inducing wrong conclusions on the presumed failure behavior. In facts, collisions reduce the statistical weight of short failures, which are collapsed in the same tuple, causing the different statistical behavior. Conversely, for $W < W^*$, we obtain a TTF distributed following a Weibull with decreasing hazard rate , (Figure 8.(c), shape parameter $= 0.74$), also in case of a small variation from W^*.

Finally, we can note that, while $MTBF_{err}$ tends to 0 at W^*, the same cannot be said for $MTTR_{err}$. The problem is that from the logs the MTTR may be underestimated since we know when the last error in the tuple manifests, but not when the failure is resolved. Calling this time difference ΔF, if we increase W we reduce the importance of ΔF with respect to the size of the tuple, hence decreasing the $MTTR_{err}$. However, increasing W affects the MTBF estimate. This suggests that *logs should be designed to register recovery events, in addition to error events*.

E. Case Study 2: System-representative Assumptions

Figure 9 reports the MTBF, MTTR, collision and truncation errors evaluated when generating logs taking into account system representative aspects. In this case logs show to be more complex and difficult to analyze. The complexity reflects in the results: in general we can observe that more collisions and truncations are caused than in the simplistic scenario, impacting negatively on MTBF and MTTR estimates. This suggests that *it is important to take into account system related aspects when validating coalescence techniques*, to better estimate the measurement errors that could be made when applying the techniques on real systems. It can also be noted that $MTBF_{err}$ and $MTTR_{err}$ become unacceptably high for *tsao*, *sliding 20m*, *sliding 5m*, and *sliding knee*, as the size of the system increases to 32768 nodes. Optimally tuned algorithms (*sliding opt*, *fixed opt*, *SC07*) and *fixed knee* seem instead to be robust against system size. However they are affected by several truncations and collisions. Hence, while $MTBF_{err}$ and $MTTR_{err}$ are low, the statistical properties of the TTF and TTR distributions are not preserved, as already observed for case 1. This suggests that *time coalescence techniques need to be rethought to be usefully adopted also in future petascale computer systems*.

From the figure it can be noted that the application of the knee rule seems to perform better as the number of nodes increases, with respect to case 1. Indeed, the presence of several overlaps smoothes the tuple count curve, where only one knee appears visible, as detailed later.

Figures 9.(c) and 9.(f) show the impact of the workload on truncations and collisions for the various algorithms. To

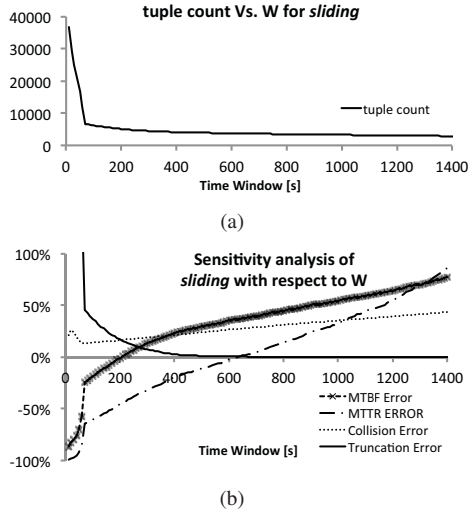

Fig. 10: Sensitivity analysis for the *sliding* algorithm in case study 2, nodes: 8192

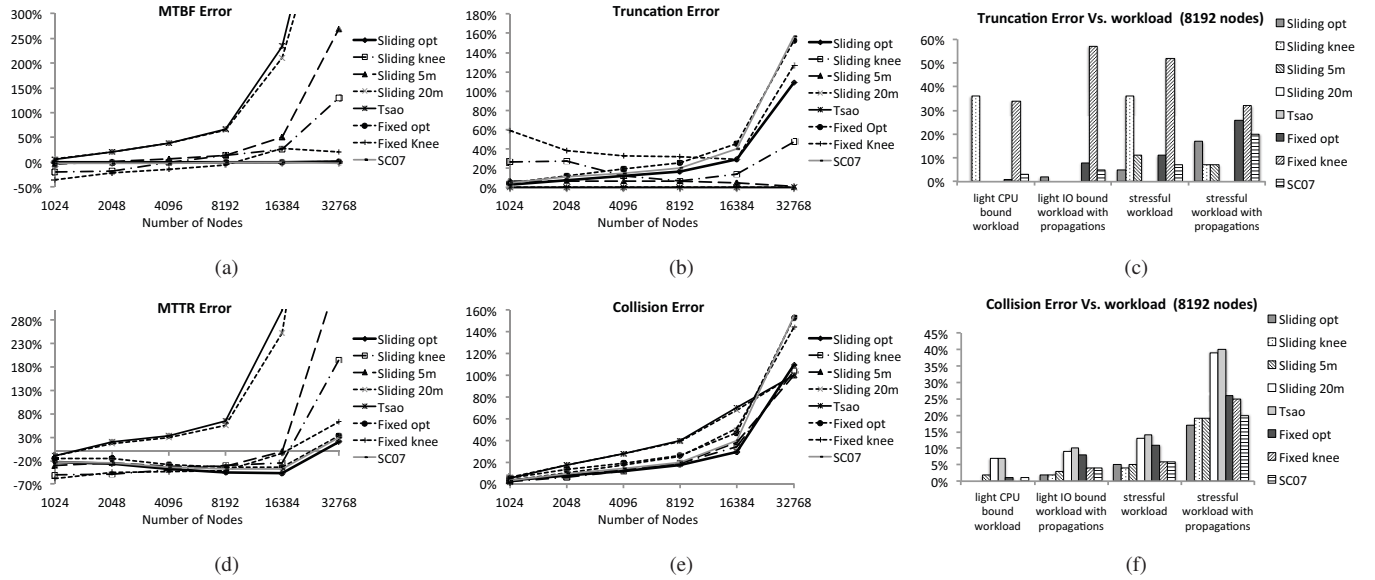

Fig. 9: Results from Case Study 2. (a) $MTBF$ and (b) $MTTR$ error, (b) truncations and (e) collisions in the considered coalescence techniques against the scale of the system; sensitivity analysis of (c) collision and (f) truncation errors for different workloads.

this aim, we repeated the experiments of the 8192 nodes case with i) a light CPU bound workload, ii) a light IO bound workload with failure propagation, iii) a stressful workload (IO bound and CPU bound jobs), and iv) a stressful workload with failure propagation. For the collisions, we can observe a similar trend for all algorithms, that is, more collisions are caused as the complexity of the workload increases. However, this result cannot be generalized for truncations, which are more dependent on the algorithm and on the specific failure process. For instance, we can note an increasing error for optimally tuned algorithms (*sliding opt*, *fixed opt*, *SC07*), but a completely workload dependent trend for the algorithms tuned with the knee rule, which again seems to perform better as the complexity of the system increases.

Figure 10 reports the results of the sensitivity analysis for the *sliding* algorithm, repeated for case study 2 for 8192 nodes. Interestingly, also in the case of more complex logs, generated with several different failure inter-arrival distributions, and considering failure propagations and a stressful workload, we can see the same trends observed for the simplistic case. Again, the time window W corresponding to the crossing point between collision and truncation curves (200 in the figure) is the one where $MTBF_{err}$ is 0. However, in this case we can notice from the tuple count curve (Figure 10.(a)) that only one knee is visible, not corresponding to the optimal window: this is due to the presence of several overlapping dynamics, which are not distinguishable as in the previous case. This suggests that *the knee rule is useful but it is difficult to apply in the case of complex and overlapping system dynamics, which should be isolated and treated separately*.

VI. Lessons learned

In this work, we proposed a novel approach and related tools to assess and validate time coalescence techniques when applied to supercomputers event logs. Experiments conducted with the proposed approach on four different coalescence techniques, and on different system configurations, revealed the following interesting findings.

- The MTBF error is strictly correlated with truncations and collisions; in particular the MTBF error is null in correspondence of a time window value such that the percentage of collisions equals the percentage of truncations. Coalescence techniques able to estimate the probability of truncations and collisions would in turn allow to define novel criteria to tune the time window, able to minimize the error on MTBF measurements.
- The statistical properties of the reconstructed time to failure distributions may be not preserved, even when optimal value for parameters are selected. The awareness of these issues, and the possibility to assess their impact on a given system, allow to estimate the expected error on measurements, which is useful to correctly interpret and weight the results.
- When the system increases in size and complexity, the presence of several overlapping dynamics can compromise the practical application of classical time coalescence techniques, which need to the rethought. A possible solution could be to isolate the different dynamics of the system (e.g., by isolating log entries related to different subsystems) and treat them separately.

We believe our approach and tools to be of interest for researchers and practitioners active in the design and development of log analysis techniques. The approach represents a valid means to validate design choices, to identify issues, and to quantitatively assess novel solutions.

Future efforts will be devoted to the extension of the approach with further mechanisms enabling the generation of informational and debug log entries and the production of realistic messages and message patterns, extracted from real systems. This will allow to extend the framework to the assessment of content-based coalescence techniques.

ACKNOWLEDGMENT

This work has been partially supported by the European Commission in the context of the CRITICAL-STEP project[4], Marie Curie IAPP action number 230672, FP7, and by the NSF grant CNS 10-18503 CISE, the Department of Energy under Award Number DE-OE0000097, the Air Force Office of Scientific Research under agreement number FA8750-11-2-0084, and Boeing corporation. The authors would like to thank Davide Bottalico for his time, expertise, and precious help in collecting and understanding the workload data from the SCOPE supercomputer.

REFERENCES

[1] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R. Sahoo. Bluegene/l failure analysis and prediction models. *DSN 2006: Int. Conference on Dependable Systems and Networks,*, pages 425–434, 2006.

[2] Bianca Schroeder and Garth A. Gibson. Understanding disk failure rates: What does an mttf of 1,000,000 hours mean to you? *IEEE Trans. Storage*, 3(3), October 2007.

[3] A. Oliner and J. Stearley. What supercomputers say: A study of five system logs. *Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP Int. Conference on*, pages 575–584, June 2007.

[4] Michael M. Tsao and Dan P. Siewiorek. Trend analysis on system error files. *In Thirteenth Annual International Symposium on Fault-Tolerant Computing*, pages 116–119, Jul 1983.

[5] J.P. Hansen and D.P. Siewiorek. Models for time coalescence in event logs. *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second Int. symp. on*, pages 221–227, Jul 1992.

[6] R. Lal and G. Choi. Error and failure analysis of a unix server. *High-Assurance Systems Engineering symp. , 1998. Proc.. Third IEEE Int.*, pages 232–239, Nov 1998.

[7] R. K. Sahoo, A. Sivasubramaniam, M. S. Squillante, and Y. Zhang. Failure data analysis of a large-scale heterogeneous server environment. In *DSN '04: Proc. of the 2004 Int. Conference on Dependable Systems and Networks*, page 772, Washington, DC, USA, 2004. IEEE Computer Society.

[8] Y. Liang, A. Sivasubramaniam, J. Moreira, Y. Zhang, R.K. Sahoo, and M. Jette. Filtering failure logs for a bluegene/l prototype. In *DSN '05: Proc. of the 2005 Int. Conference on Dependable Systems and Networks*, pages 476–485, Washington, DC, USA, 2005. IEEE Computer Society.

[9] A. Thakur and R.K. Iyer. Analyze-now-an environment for collection and analysis of failures in a network of workstations. *Reliability, IEEE Transactions on*, 45(4):561–570, Dec 1996.

[10] M. Kalyanakrishnam, Z. Kalbarczyk, and R. Iyer. Failure data analysis of a lan of windows nt based computers. In *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems SRDS 99*, pages 178 –187, 1999.

[11] A. Ganapathi and D. Patterson. Crash data collection: A windows case study. In *DSN '05: Proc. of the 2005 Int. Conference on Dependable Systems and Networks*, pages 280–285, Washington, DC, USA, 2005. IEEE Computer Society.

[12] C. Simache, M. Kaâniche, and A. Saidane. Event log based dependability analysis of windows nt and 2k systems. In *PRDC '02: Proc. of the 2002 Pacific Rim Int. symp. on Dependable Computing*, page 311, Washington, DC, USA, 2002. IEEE Computer Society.

[13] J. Xu, Z. Kalbarczyk, and R. K. Iyer. Networked windows nt system field failure data analysis. In *PRDC '99: Proc. of the 1999 Pacific Rim Int. symp. on Dependable Computing*, page 178, Washington, DC, USA, 1999. IEEE Computer Society.

[14] J. Stearley and A.J. Oliner. Bad words: Finding faults in spirit's syslogs. In *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, pages 765 –770, may 2008.

[15] D.D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J.M. Doyle, W.H. Sanders, and P.G. Webster. The mobius framework and its implementation. *Software Engineering, IEEE Transactions on*, 28(10):956 – 969, oct 2002.

[16] Chinghway Lim, N. Singh, and S. Yajnik. A log mining approach to failure analysis of enterprise telephony systems. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 398 –403, june 2008.

[17] C. Di Martino, D. Cotroneo, Z. Kalbarczyk, and R. K. Iyer. A framework for assessing the dependability of supercomputers via automated log analysis. In *Sup. volume of Proc. of the Int. Conference on Dependable Systems and Networks, Anchorage, AK.*, pages 383–384, 2008.

[18] Thomas J. Hacker, Fabian Romero, and Christopher D. Carothers. An analysis of clustered failures on large supercomputing systems. *Journal of Parallel and Distributed Computing*, 69(7):652 – 665, 2009.

[19] J. Brevik, D. Nurmi, and R. Wolski. Automatic methods for predicting machine availability in desktop grid and peer-to-peer systems. In *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '04, pages 190–199, Washington, DC, USA, 2004. IEEE Computer Society.

[20] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance computing systems. In *DSN '06: Proc. of the Int. Conference on Dependable Systems and Networks*, pages 249–258, Washington, DC, USA, 2006. IEEE Computer Society.

[21] Taliver Heath, Richard P. Martin, and Thu D. Nguyen. Improving cluster availability using workstation validation. In *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '02, pages 217–227, New York, NY, USA, 2002. ACM.

[22] Thomas J. Hacker and Zdzislaw Meglicki. Using queue structures to improve job reliability. In *Proceedings of the 16th international symposium on High performance distributed computing*, HPDC '07, pages 43–54, New York, NY, USA, 2007. ACM.

[23] S. Fu and C. Xu. Exploring event correlation for failure prediction in coalitions of clusters. In *SC '07: Proc. of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–12. ACM, 2007.

[24] D. Tang and R.K. Iyer. Analysis and modeling of correlated failures in multicomputer systems. *Computers, IEEE Transactions on*, 41(5):567 –577, may 1992.

[25] X. Castillo and D.P. Siewiorek. Workload, performance, and reliability of digital computing systems. *Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years'., Twenty-Fifth Int. symp. on*, pages 367–, Jun 1995.

[26] T. T. Y. Lin and D. P. Siewiorek. Error log analysis: statistical modeling and heuristic trend analysis. *Reliability, IEEE Transactions on*, 39(4):419–432, 1990.

[27] Ramendra K. Sahoo, Anand Sivasubramaniam, Mark S. Squillante, and Yanyong Zhang. Failure data analysis of a large-scale heterogeneous server environment. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, pages 772–, Washington, DC, USA, 2004. IEEE Computer Society.

[28] Antonio Pecchia, Domenico Cotroneo, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. Improving log-based field failure data analysis of multi-node computing systems. *Dependable Systems and Networks, International Conference on*, 0:97–108, 2011.

[29] R. Vaarandi. Mining event logs with slct and loghound. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 1071 –1074, april 2008.

[30] Ziming Zheng, Zhiling Lan, B.H. Park, and A. Geist. System log pre-processing to improve failure prediction. In *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, pages 572 –577, 29 2009-july 2 2009.

[31] M. F. Buckley and D. P. Siewiorek. A comparative analysis of event tupling schemes. In *FTCS '96: Proc. of the The Twenty-Sixth Annual Int. symp. on Fault-Tolerant Computing (FTCS '96)*, page 294, Washington, DC, USA, 1996. IEEE Computer Society.

[32] Ziming Zheng, Li Yu, Wei Tang, Zhiling Lan, Rinku Gupta, Narayan Desai, Susan Coghlan, and Daniel Buettner. Co-analysis of ras log and job log on blue gene/p. In *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, IPDPS '11, pages 840–851, Washington, DC, USA, 2011. IEEE Computer Society.

[33] I. Lee and R. K. Iyer. Software dependability in the tandem guardian system. *IEEE Trans. Softw. Eng.*, 21(5):455–467, 1995.

[34] Xin Li, Kai Shen, Michael C. Huang, and Lingkun Chu. A memory soft error measurement on production systems. In *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, pages 21:1–21:6, Berkeley, CA, USA, 2007. USENIX Association.

[35] Francesco Palmieri and Silvio Pardi. Towards a federated metropolitan area grid environment: The scope network-aware infrastructure. *Future Generation Computer Systems*, 26(8):1241 – 1256, 2010.

[36] Francesco Palmieri, Silvio Pardi, and Paolo Veronesi. A fault avoidance strategy improving the reliability of the egi production grid infrastructure. In *Proceedings of the 14th international conference on Principles of distributed systems*, OPODIS'10, pages 159–172, Berlin, Heidelberg, 2010. Springer-Verlag.

[4]http: //www.critical-step.eu