Tracking in a Spaghetti Bowl: Monitoring Transactions Using Footprints

Animashree Anandkumar*
Electrical & Computer Engr.,
Cornell University,
Ithaca, NY 14853
aa332@cornell.edu

Chatschik Bisdikian
Networking Technologies,
IBM Watson Research,
Hawthorne, NY 10532
bisdik@us.ibm.com

Dakshi Agrawal
Networking Technologies,
IBM Watson Research,
Hawthorne, NY 10532
agrawal@us.ibm.com

ABSTRACT

The problem of tracking end-to-end service-level transactions in the absence of instrumentation support is considered. The transaction instances progress through a state-transition model and generate time-stamped footprints on entering each state in the model. The goal is to track individual transactions using these footprints even when the footprints may not contain any tokens uniquely identifying the transaction instances that generated them. Assuming a semi-Markov process model for state transitions, the transaction instances are tracked probabilistically by matching them to the available footprints according to the maximum likelihood (ML) criterion. Under the ML-rule, for a two-state system, it is shown that the probability that all the instances are matched correctly is minimized when the transition times are i.i.d. exponentially distributed. When the transition times are i.i.d. distributed, the ML-rule reduces to a minimum weight bipartite matching and reduces further to a first-in first-out match for a special class of distributions. For a multi-state model with an acyclic state transition digraph, a constructive proof shows that the ML-rule reduces to splicing the results of independent matching of many bipartite systems.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Distributed systems, Performance evaluation

General Terms

Performance, Theory

Keywords

Transaction monitoring, maximum-likelihood tracking, semi-Markov process, bipartite matching.

*Major part of this work was performed during A. Anandkumar's internship at IBM Research. She is also supported in part by the Army Research Office under Grant ARO-W911NF-06-1-0346.

SIGMETRICS'08, June 2–6, 2008, Annapolis, Maryland, USA.

1. INTRODUCTION

Imagine a small discount brokerage firm that distinguishes itself by providing its customers a web-portal that can be "personalized". As competitors catch up, it keeps itself ahead of the pack by offering a plethora of new options to enhance the user-experience. It grows rapidly over a five-year period as its IT staff struggles to keep pace with the rapid growth of the customer base as well as growth in the number of applications that run behind the scenes to provide new utilities and options to its customers. IT infrastructure that started as a collection of ten-twenty servers laid down neatly in a tiered architecture becomes a *spaghetti bowl* of five hundred servers without a clear picture of how the response time or the availability of a behind-the-scenes application impacts the response time of a utility or an option exposed to the customer. Worse still, if customers complain about aborted or time-consuming transactions, then the IT staff lacks visibility for efficient troubleshooting. It is not unusual for problem resolutions to take days of manual debugging.

In an ideal scenario, as the brokerage firm grew, its IT staff would adhere to the best practices such as those advocated by the ITIL specifications to keep track of its IT infrastructure. It would use industry standards such as the open-group ARM instrumentation [1] to generate transaction correlators or *tokens* that may be used to track the flow of transactions. However, that rarely is the case — often what one ends up with is an IT infrastructure that is at best partly documented and partly instrumented. The challenge then is to provide *end-to-end monitoring* of transactions in such environments with minimal expense and disruption to the operations. We use the term transaction not in the sense of a backend database update [8], but an end-to-end business operation spanning multiple applications, servers, middleware and backend databases, e.g., an entire online trading transaction with login, order creation, payment, etc.

A solution to end-to-end transaction monitoring will comprise of three pieces: (a) discovery of IT artifacts, such as servers and applications, on which the transaction depends, (b) modeling of relationships among these IT artifacts in the context of the transaction, and (c) monitoring of IT artifacts to draw conclusions regarding the status of a transaction. Each of these pieces will pose different challenges depending on the degree of information and instrumentation available in the system. For example, discovery and modeling can be partly accomplished by tapping into and examining the network flows, by correlating the message headers in messaging middleware, or by looking at the J2EE EJB relationships. The study presented in this paper is part of a

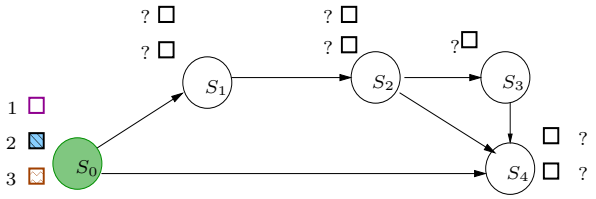


Figure 1: Tracking transactions in state-transition model using footprints without token information.

broader activity developing for each of the above pieces in a variety of IT environments, where there is access only to the time-stamped *transaction footprints*. A footprint is generated when a transaction is processed by an application and is recorded in the application logs. The goal is to find fundamental limits on the performance of methods that use such footprints to track and monitor transactions in a figurative spaghetti bowl described above.

If *all* the footprints left by a transaction contain *tokens*, i.e., an identifier unique to the transaction, then tracking the transaction is simple. Unfortunately, only a small number of footprints may contain tokens, e.g., in Fig.1, other than the footprints at state S_0, none of the other footprints contain tokens. In such cases, it may not be possible to identify the unique source of each footprint with certainty, except for simple cases such as a strictly ordered process scheduling like first-in-first-out (FIFO) or last-in-first-out (LIFO). In general, parallel processing of transactions leads to random ordering of transaction operations in distributed and multi-threaded environments. Hence, there is uncertainty in tracking transaction instances, leading to *maximum-likelihood* (ML) estimates.

These issues lead to a range of questions: can we provide bounds or guarantees on the probability of correctly tracking the transaction instances as they pass through the IT infrastructure? Which transaction models maximally confound the tracking system; in other words, how does the state model of a transaction affect the tracking probability? Can the algorithms for tracking transactions be implemented in a decentralized manner with reasonable complexity? If tokens are present in footprints generated at certain states, then what is the improvement in the tracking precision?

1.1 Technical Approach

We model the progress of a transaction through the IT infrastructure using a state-transition model. A transaction generates a footprint on entering each state in the model, stamped with the time of entry. The state-transition model has a unique start state, denoted by S_0, where each transaction starts, and a set of end states, in one of which, each transaction ultimately ends. An example of state-transition digraph is shown in Fig.1 with start state S_0 and end state S_4. There are three footprints in S_0 indicating that three transactions entered the system, while the two footprints in S_4 indicate that two transactions have left the system. We assume that the state transitions follow a Markov chain, and additionally, that the time spent by a transaction instance at a state is only dependent on the current and the next state to which it transitions. This model is applicable when the time taken for a transaction to execute an application (represented as a state in the model) is only dependent on its outcome (represented as another state) and not on the past history of the transaction. Formally, such a model is known as a semi-Markov process, defined in Section 2.1.

By tracking a set of transaction instances, we mean finding the most likely sequence of states visited by each of these transaction instances and the estimates of the times spent in these states by each instance. When only footprints are available, this reduces to matching footprints at various states to the individual transaction instances that generated them. We take a probabilistic approach by incorporating the available (statistical) information about transition times between different states. In our approach, optimal tracking refers to using the maximum-likelihood (ML) rule that maximizes the probability that all the footprints are matched correctly to the transactions that generated them.

1.2 Contributions

In general, optimal tracking involves searching over all possible correspondences between footprints and transaction instances, which can be exponential in the number of footprints. The goal of this paper is to find tractable polynomial-time solutions for special cases of the system model. The contribution of this work is four-fold: firstly, we show through a constructive proof that optimal tracking for a multi-state acyclic semi-Markov process (SMP) without any tokens can be done by splicing the results of independent optimal tracking in a series of bipartite graphs or two-state systems, each consisting of a "high-level" start and end state. Hence, the complexity of optimal tracking is determined by the number of such bipartite systems and when the state-transition digraph of the SMP is a tree, we show that it is the number of non-terminating states in the tree.

Secondly, we show that for a two-state system, under i.i.d. transition times, optimal ML-tracking reduces to a minimum-weight perfect matching. We further compare the performance of optimal tracking with the simple FIFO rule, which matches footprints according to the order of their timestamps, and provide a class of transition-time distributions for which the two rules coincide. We also analyze the effect of footprints arriving in the correct/incorrect temporal order at the monitoring engine.

Thirdly, we derive worst-case performance bounds for optimal ML-tracking in two-state systems. We show that ML-tracking has the worst performance when the transition times of the transactions are drawn from i.i.d. uniform or exponential distributions, for the case when all the transactions have completed their operations and exited the system. On the other hand, when some transactions are still resident in the two-state system, only the i.i.d. exponential distribution has the worst performance. In both the above cases, the worst-case ML-probability that all the transactions are matched correctly to their footprints is equal to the reciprocal of the number of unique valid matches between them.

Lastly, we consider the case when each footprint at certain model states contains a token, unique to the transaction generating it. For the special case of a linear SMP with tokens available at the start and the terminating state, we show that the ML-tracking can still be decentralized to a series of bipartite matchings, as in the tokenless case. Here, the ML-rule first finds the matches among all the tokenless footprints to form the most-likely intermediate paths, and then jointly matches the tokenized footprints to these inter-

mediate paths. When the problem is extended to the case where the state transition digraph is a tree, with only certain terminating states tokenized, the ML-rule reduces to a transportation problem with exclusionary side constraints, an NP-hard problem, and not to bipartite matching.

Taken together, these contributions provide a foundation on which an efficient tracking solution can be built. To the best of our knowledge, there are only few prior works focusing on tracking concurrent individual transactions (see Section 7), and we present its first careful study here. We consider the use of a semi-Markovian model with i.i.d. transitions and (mostly) tokenless footprints as both a contribution and a limitation. On one hand, this model leads to a considerable simplification of the optimal ML-tracking into a series of bipartite matchings. Further, this model serves as a benchmark for tracking performance, since a tracking system is maximally confounded. On the other hand, under such extreme assumptions, even optimal tracking with acceptable accuracy is only possible when the number of interleaving instances and model states are small.

The paper is organized as follows. In Section 2, we provide a formal definition of the system model and optimal tracking. In Section 3, we consider tracking in two-state systems and then consider a general SMP in Section 4, and finally tracking with tokens in Section 5. Section 6 has the simulations results, Section 7 has the related work and Section 8 concludes the paper.

2. SYSTEM MODEL

2.1 Notations and Definitions

Let $f_X(x)$ be the probability density function (pdf) of a continuous random variable X and $\bar{F}_X(x) := P[X > x]$ its complementary cumulative distribution function (ccdf). For a matrix $\mathbf{A}$, let $A(i,j)$ denote the element in its i^{th} row and j^{th} column. Let $\boldsymbol{\pi}$ denote a permutation vector over $\{1,\ldots,n\}$, $\log x$, the natural logarithm of x and, $|A|$, the cardinality of a set A. For sets A and B, let $A \backslash B = \{i : i \in A, i \notin B\}$.

For an undirected graph $G = (V, E)$, let (i,j) denote the edge between i and j and $\deg(i)$ be the degree of node i. For a bipartite graph $G = (V_0 \cup V_1, E)$, the edges are represented by a 0-1 biadjacency matrix $\mathbf{A} = [A(i,j)]$, where $A(i,j) = 1$ indicates an edge between $V_0(i)$ and $V_1(j)$. A matching $M \subset E$ is a set of pairwise non-adjacent edges, i.e., no two edges share a common vertex. A maximum cardinality matching is a matching that contains the largest possible number of edges. A perfect matching is a matching where there is no unmatched vertex and a minimum weight perfect matching minimizes the sum of the matched edge weights. For a directed graph (digraph), when there is an edge from i to j, j is an immediate successor of i, and i an immediate predecessor of j. The set of all immediate successors of i is denoted by $\mathcal{N}(i)$, and the set of all immediate predecessors by $\mathcal{P}(j)$. A directed acyclic graph (DAG) has at least one source (start state) with no incoming edges and at least one sink (end state) with no outgoing edges.

We now define a semi-Markov process (SMP) [14]. Let $S_i, i = 0,\ldots,N_s$ denote the i^{th} state of the process and let $T_{i,j}$ denote the (random) time to transition from state S_i to state S_j. A process is said to be semi-Markov if the sequence of states visited is a Markov chain, with transition probability matrix denoted by $\mathbf{P} = [P(i,j)]$, and each transition time $T_{i,j}$ is a random variable that depends only on the states S_i and S_j involved in the transition. We assume that each transaction progresses through the system according to a general SMP with each transition time $T_{i,j}$ drawn from a known pdf $f_{T_{i,j}}$ having a continuous interval as its support.

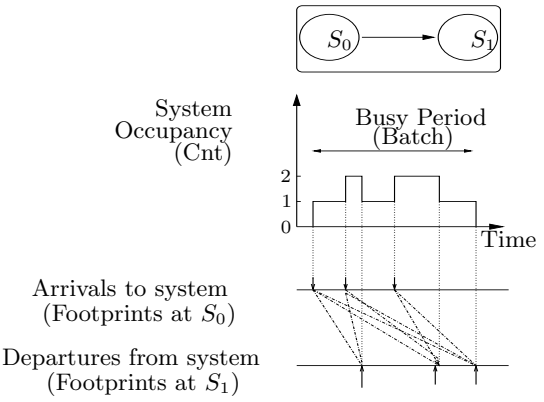


Figure 2: Time stamps of footprints and possible transitions in 2-state system.

We consider a connected state transition digraph for the SMP, since otherwise transactions in different components can be monitored independently. We make a simplifying assumption that the state transition digraph is acyclic (DAG). This ensures that all the transactions are processed in one direction, and that no transaction can leave more than one footprint at a state or be present in the system forever. DAG is a generalization of the tree in which certain subtrees can be shared by different parts of the tree.

We also assume that multiple transactions in the system progress independent of one another, an extreme scenario where the tracking system is maximally confounded, and this may be a valid approximation when the transactions are processed concurrently. This implies that the transition times are independent of the system load. Typically, load-dependent transition times occur due to buffering of the incoming transactions and buffered transactions are usually processed in a fixed order, e.g., $M/M/1$ queue. Tracking transactions in such queueing systems is straight-forward, and hence, we do not consider such systems.

2.2 Optimal Tracking (ML-Rule)

A footprint is defined as a timestamped entry created in the application log when a transaction enters a state in the model. Since we track the transactions using these timestamps, we need to address the issue of synchronization between the clocks in different servers of the system. There are many solutions to achieve synchronization in distributed systems, e.g., network time protocol (NTP). Even when such a solution is not implemented, for most applications, the individual requests making up a transaction have sojourn times lasting several tens-hundreds of milli-seconds per component, an order of magnitude larger than the synchronization error. Moreover, whenever tokens are present, we can minimize the effect of asynchrony through matching requests across components/tiers, thereby preventing the accumulation of synchronization errors. Hence, the effects of asynchrony on tracking performance may not be significant.

In addition to the timestamp, a footprint may optionally contain a unique identifier or a token that ties it to the transaction instance. By convention, the footprints at the (unique) start state S_0 are each assigned a token. We assume that no footprint is missing from the log records. We consider the general case where at the time of observation, transaction instances are still residing at different states of the system, and hence, all the footprints that these transactions will eventually generate are not yet available. Tracking transactions in such cases is affected by the assumption that the records appear in logs as soon as they are written by the applications, i.e., the write is not buffered, ensuring that the records from the different logs arrive at a monitoring engine in the correct temporal order. We consider both the scenarios when the monitoring engine may or may not receive the footprints in correct order.

Any valid match between the footprints and the transactions cannot have two transactions sharing a common footprint, since each footprint is generated by a unique transaction. Hence, any valid match can be represented by the set of permutation vectors $\boldsymbol{\pi}_k$, for each state S_k in the model. Let $\mathbf{Y}_k$ be the vector of the timestamps of the footprints at state S_k. When the monitoring engine receives the footprints in the correct temporal order, $\mathbf{Y}_k$ is in ascending order with the most recent footprint being the last entry. Let $\mathbf{Y}_k^{\boldsymbol{\pi}_k}$ be the permutation of $\mathbf{Y}_k$, according to the permutation vector $\boldsymbol{\pi}_k$. By convention, we assign tokens to footprints $\mathbf{Y}_0$ at the start state S_0, and hence, the permutation vector at the start states S_0 is set to identity ($\boldsymbol{\pi}_0 = \mathbf{I}$). In other words, we find the correspondence of all other footprints in the system with respect to the footprints at S_0.

When the joint pdf of the transaction transition times $f_{\mathbf{T}}$ is known, we can quantify the tracking performance as the probability that all the transaction instances are matched correctly to their footprints and this is maximized by the maximum-likelihood (ML) rule. Hence, for $N_s + 1$ number of states, the optimal ML-tracking of transactions reduces to finding a set of N_s number of permutation vectors,

$$[\hat{\boldsymbol{\pi}}_1^{\mathrm{ML}}, \ldots, \hat{\boldsymbol{\pi}}_{N_s}^{\mathrm{ML}}] := \arg\max_{\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_{N_s}} \mathbb{P}(\mathbf{Y}_1^{\boldsymbol{\pi}_1}, \ldots, \mathbf{Y}_{N_s}^{\boldsymbol{\pi}_{N_s}} | \mathbf{Y}_0). \quad (1)$$

In general, solving (1) is NP-hard. The rest of the paper primarily deals with the special cases, starting with the two-state system, where (1) can be solved efficiently.

3. TWO-STATE SYSTEM

We consider a two-state model, which will serve as a foundation for more elaborate models. For this model we will show that the ML-rule reduces to a perfect matching in a bipartite graph under i.i.d. transition times.

3.1 Preliminaries

Fig.2 shows transactions in a two state model with start state S_0 and end state S_1. Fig.3 shows the system representation, where the footprints $\mathbf{Y}_0$ and $\mathbf{Y}_1$ at states S_0 and S_1 are related through an unknown permutation vector $\boldsymbol{\pi}$,

$$Y_0(j) = Y_1(\pi(j)) - T(j), \quad 1 \le j \le |\mathbf{Y}_1|, \quad (2)$$

where $T(j)$ is the transition time of $Y_0(j)$, the j^{th} footprint at S_0, $Y_1(\pi(j))$ is the j^{th} element of the permuted $\mathbf{Y}_1$, according to $\boldsymbol{\pi}$. The footprints $Y_0(j)$ and $Y_1(\pi(j))$ are generated by the same transaction. We have $|\mathbf{Y}_0| \le |\mathbf{Y}_1|$, since some transactions may still be resident at S_0. The number of instances in state S_0 at the time of observation is

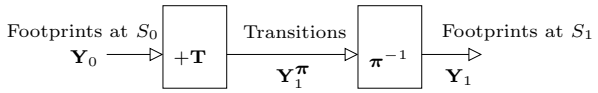


Figure 3: Given $\mathbf{Y}_0, \mathbf{Y}_1$, find permutation $\boldsymbol{\pi}$.

$$\mathrm{Cnt}(S_0) = |\mathbf{Y}_0| - |\mathbf{Y}_1|. \quad (3)$$

A *batch* of footprints at S_0 and S_1 starts when $\mathrm{Cnt}(S_0) = 0$ and is said to be *complete* when $\mathrm{Cnt}(S_0) = 0$ again, i.e., all the transactions have exited the system; otherwise the batch is *partial*. It is easy to see that a valid match can occur only between the footprints in the same batch and not across batches. Henceforth, $(\mathbf{Y}_0, \mathbf{Y}_1)$ denote the set of footprints in a single batch, unless otherwise mentioned.

We first consider the case of non-parametric transition times, where transition-time pdf is unknown. We know that the lower bound of the pdf support is zero due to causality. Optionally, we may know the upper bound Δ of the pdf support, i.e., $0 \le T \le \Delta$; it is known as the *deadline* and is infinite, if not specified. Due to the presence of timestamps, some matches $\boldsymbol{\pi}$ may violate the causality and the deadline constraints. In order to find the number of valid matches, define a bipartite graph $G = (V_0 \cup V_1, E)$, where $V_i(j)$ represents the j^{th} footprint at state S_i and the edges in E represent the valid correspondences between the footprints at S_0 and S_1. Hence, an edge is added to E, whenever the difference between the timestamps of the two footprints satisfies the pdf support bounds. Since the actual pdf of transition times is unknown, the edges are unweighted. Given the biadjacency matrix $\mathbf{A}$, the number of *unique* valid matches in a batch, denoted by $N_B(|\mathbf{Y}_0|, |\mathbf{Y}_1|, \mathbf{A})$, provides an idea of the precision of tracking individual instances. For a complete batch with $|\mathbf{Y}_0| = |\mathbf{Y}_1| = n$, $N_B(n, n, \mathbf{A})$ is given by the permanent of the biadjacency matrix, $\mathrm{perm}(\mathbf{A})$,

$$N_B(n, n, \mathbf{A}) = \mathrm{perm}(\mathbf{A}) := \sum_{\boldsymbol{\pi}} \prod_{i=1}^{n} A(i, \pi(i)), \quad (4)$$

where the sum is over all the permutation vectors $\boldsymbol{\pi}$ over $\{1, \ldots, n\}$. Hence, the timestamps in the footprints reduce the number of valid matches. Since there is at least one perfect match, corresponding to the true transition pattern, we have $1 \le \mathrm{perm}(\mathbf{A}) \le n!$. The upper bound is achieved for a complete bipartite graph, i.e., when all the instance departures from S_0 occur after all the arrivals in the batch.

For a partial batch, some of the footprints at S_1 are not yet generated, and hence, a perfect bipartite matching is not feasible. For the case when the footprints may not arrive in the correct temporal order, any maximum cardinality bipartite matching is a valid match. However, when the footprints arrive in the correct order, we have additional information about the transactions which are still resident at S_0 and this changes the structure of the bipartite graph. Given that $|\mathbf{Y}_1| = k < |\mathbf{Y}_0| = n$, there are $n - k$ number of instances that have not yet made the transition and their departures

occur after time $Y_1(k)$, the timestamp of the most recent footprint at S_1. This information is incorporated by adding $n-k$ number of identical copies of a new (dummy) node, denoted by $V_1(\delta)$, to the bipartition V_1. Edges are added between $V_1(\delta)$ and any node $V_0(i)$ if $Y_1(k) - Y_0(i) < \Delta$, i.e., the deadline has not yet passed. Since all the dummy nodes are identical, some of the perfect matchings in this bipartite graph are now equivalent, and the number of unique matchings in a partial batch is

$$N_B(n,k,\mathbf{A}) = \frac{\text{perm}(\mathbf{A})}{(n-k)!}, \tag{5}$$

since the permutations among the copies of the added node $V_1(\delta)$ are equivalent. When $n=k$, it reduces to (4).

It is NP-hard to compute perm($\mathbf{A}$) in (5). Hence, we resort to approximations and bounds [23, 30]. In [5, A.1], we provide these bounds and also show that the bipartite graph of a footprint batch is *elementary*, where every edge in the bipartite graph occurs in at least one perfect matching. Hence, the creation of batches avoids adding unnecessary edges, leading to a sparse construction and the matchings can be done independently across different batches.

For non-parametric matching, without the knowledge of transition-time pdf, we use the first-in first-out rule (FIFO) to obtain a maximum cardinality match. The FIFO rule matches each footprint in V_1 with the earliest unmatched footprint in V_0, having an edge with it. It provides a maximum match for any convex bipartite graph [18] and the graph here is actually biconvex, since we assume that the transition time T take values in a continuous interval. The FIFO-rule runs in $O(n)$ time, for a n-batch, when the footprints arrive in the correct temporal order; otherwise, we first order the footprints and then apply the FIFO rule.

3.2 Optimal Tracking

When the joint pdf $f_{\mathbf{T}}$ of the transaction transition times $\mathbf{T} = [T(j)]$ is known in a two-state system, the most-likely (ML) match in (1) reduces to

$$\begin{aligned} \hat{\boldsymbol{\pi}}^{\text{ML}}(\mathbf{Y}_0, \mathbf{Y}_1; f) &:= \arg\max_{\boldsymbol{\pi}} \mathbb{P}(\mathbf{Y}_1^{\boldsymbol{\pi}} | \mathbf{Y}_0) && (6) \\ &= \arg\max_{\boldsymbol{\pi}} f_{\mathbf{T}}[\mathbf{Y}_1^{\boldsymbol{\pi}} - \mathbf{Y}_0], && (7) \end{aligned}$$

for a complete batch of n footprints, $|\mathbf{Y}_0| = |\mathbf{Y}_1| = n$. The probability that all the n footprints in the batch are matched correctly is maximized under the ML-rule, and is

$$P^{\text{ML}}(\mathbf{Y}_0, \mathbf{Y}_1; f) = \frac{f_{\mathbf{T}}[\mathbf{Y}_1^{\hat{\boldsymbol{\pi}}^{\text{ML}}} - \mathbf{Y}_0]}{\sum_{\boldsymbol{\pi}} f_{\mathbf{T}}[\mathbf{Y}_1^{\boldsymbol{\pi}} - \mathbf{Y}_0]}, \tag{8}$$

where the likelihood of the ML-permutation $\hat{\boldsymbol{\pi}}^{\text{ML}}$ is normalized by the sum of likelihoods of all valid permutations $\boldsymbol{\pi}$. Since at least one permutation has positive likelihood (the true pattern), the denominator in (8) is strictly positive.

For a partial batch $(\mathbf{Y}_0, \mathbf{Y}_1)$ with $|\mathbf{Y}_0| = n$ and $|\mathbf{Y}_1| = k < n$, when the footprints arrive in the correct order at the monitoring engine, we have additional information about the instances that have not yet made the transition. Formally, let $\mathcal{A}_f^{\boldsymbol{\pi}}(\mathbf{Y}_0, \mathbf{Y}_1)$ be the event that for $1 \le \pi(i) \le k$, the $\pi(i)^{\text{th}}$ footprint at S_1 is generated by the i^{th} instance at S_0 and for $k < \pi(i) \le n$, the i^{th} instance is still residing at S_0,

$$\begin{aligned} \mathcal{A}_f^{\boldsymbol{\pi}}(\mathbf{Y}_0, \mathbf{Y}_1) \;:=\; [&\bigcap_{1 \le \pi(i) \le k} T(i) \in \mathcal{I}(Y_1(\pi(i)) - Y_0(i)) \\ &\bigcap_{k < \pi(i) \le n} T(i) > Y_1(k) - Y_0(i)], \end{aligned} \tag{9}$$

where $\mathcal{I}(t)$ denotes the infinitesimal interval $[t, t+dt]$ and $Y_1(k)$ is the most recent footprint at S_1. The ML-rule for a partial batch is

$$\hat{\boldsymbol{\pi}}^{\text{ML}}(\mathbf{Y}_0, \mathbf{Y}_1; f) = \arg\max_{\boldsymbol{\pi}} \mathbb{P}[\mathcal{A}_f^{\boldsymbol{\pi}}(\mathbf{Y}_0, \mathbf{Y}_1)],$$

and the probability $P^{\text{ML}}(\mathbf{Y}_0, \mathbf{Y}_1; f)$ is

$$P^{\text{ML}}(\mathbf{Y}_0, \mathbf{Y}_1; f) = \frac{\mathbb{P}[\mathcal{A}_f^{\hat{\boldsymbol{\pi}}^{\text{ML}}}(\mathbf{Y}_0, \mathbf{Y}_1)]}{\sum_{\boldsymbol{\pi}} \mathbb{P}[\mathcal{A}_f^{\boldsymbol{\pi}}(\mathbf{Y}_0, \mathbf{Y}_1)]}. \tag{10}$$

When $n = k$, the batch is complete and $\mathbb{P}[\mathcal{A}_f^{\boldsymbol{\pi}}(\mathbf{Y}_0, \mathbf{Y}_1)] = f_{\mathbf{T}}[\mathbf{Y}_1^{\boldsymbol{\pi}} - \mathbf{Y}_0] d\mathbf{t}$ and hence, (10) reduces to (8).

In general, finding the ML-rule in (10) and (8) poses a significant computational challenge and we address it in the next four subsections. We derive the performance bounds of the ML-based tracking probability for arbitrary joint-pdfs of the transition times and study the rule under i.i.d. transition times. We then compare the ML-matching to the *distribution-free* FIFO matching and provide conditions when they coincide.

3.2.1 Reduction of ML-rule to Weighted Matching

The ML-rule for a general joint-pdf $f_{\mathbf{T}}$ of the transition times $\mathbf{T} = [T(j)]$ requires search over all the permutation vectors $\boldsymbol{\pi}$, which could be exponential in the batch size. We now make a simplifying assumption that all the instance transition times $T(1), T(2), \ldots$ are i.i.d. with pdf f_T. For a $(\mathbf{Y}_0, \mathbf{Y}_1)$ batch, the ML-rule now reduces to

$$\begin{aligned} \hat{\boldsymbol{\pi}}^{\text{ML}}(\mathbf{Y}_0, \mathbf{Y}_1; f) = \arg\max_{\boldsymbol{\pi}} &\prod_{1 \le \pi(i) \le k} f_T[Y_1(\pi(i)) - Y_0(i)] \\ &\prod_{k < \pi(i) \le n} \bar{F}_T[Y_1(k) - Y_0(i)], \end{aligned} \tag{11}$$

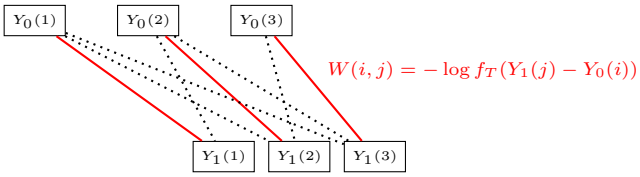
where $\bar{F}_T(t) = \mathbb{P}[T > t]$ is the complementary cumulative distribution function (ccdf). For the bipartite graph $G = (V_0 \cup V_1, E)$, defined in the previous section with the added node $V_1(\delta)$ (henceforth, known as the ccdf node), we now assign a weight $W(i,j)$, for each edge (i,j),

$$W(i,j) := \begin{cases} -\log f_T[Y_1(j) - Y_0(i)], & j \le k \quad (12a) \\ -\log \bar{F}_T[Y_1(k) - Y_0(i)], & j = \delta. \quad (12b) \end{cases}$$

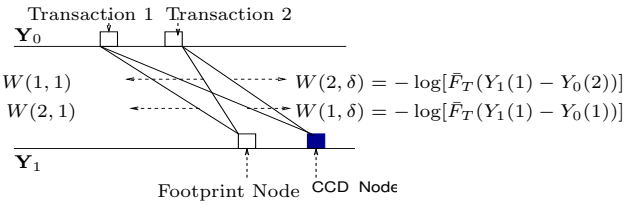
The ccdf node is added to the bipartite graph on the assumption that the footprints arrive in the correct order. When instead, the footprints do not arrive in order, the ccdf node is not added and the edge weights are solely given by (12a). We will compare the performances of the two scenarios through simulations in Section 6.

A minimum-weight perfect matching is given by

$$\boldsymbol{\pi}^*(n,k,\mathbf{W}) := \arg\min_{\boldsymbol{\pi}} \sum_{i=1}^{n} W(i, \pi(i)), \tag{13}$$



(a) Matching a complete batch.



(b) Matching a partial batch with ccdf events.

Figure 4: Reduction of ML-rule for i.i.d. transitions to minimum-weight perfect matching.

and let W^* be the value of the minimum weight match

$$W^* := \min_{\boldsymbol{\pi}} \sum_{i=1}^{n} W(i, \pi(i)). \tag{14}$$

An example of minimum weight matching is shown in Fig.4a for a complete batch and Fig.4b for a partial batch. A minimum weight perfect matching can be performed in $O(n(|E|+n\log n))$ for a n-batch and $|E|$ number of edges via the Hungarian algorithm [18]. Hence, we see that the creation of batches leads to efficient implementation, since n and $|E|$ are substantially reduced. In the lemma below, we provide the ML-rule $\boldsymbol{\pi}^{\text{ML}}$ and the matching probability P^{ML} .

LEMMA 1 (I.I.D. TRANSITIONS). *In a two-state system, for i.i.d. transaction transition times according to a given pdf f_T, and the footprints arriving in the correct order, the ML rule is given by the minimum weight perfect matching in (13) and the probability that all footprints in an $(n, k, \mathbf{W})$ batch are matched correctly under the ML-rule is*

$$P^{ML}(n, k, \mathbf{W}) = \frac{(n-k)!\exp(-W^*)}{perm[\exp(-\mathbf{W})]}, \tag{15}$$

where $\exp(-\mathbf{W}) = [\exp(-W(i,j))]$, $\mathbf{W}$ is given by (12), and W^* by (14). For the case when the footprints do not arrive in the correct order, the ML-rule is a minimum weight maximum cardinality, based solely on the edge weights in (12a).

Proof: From (10) and (11). □

Hence, we see that the complexity of the ML-rule is significantly reduced for i.i.d. transitions. Additionally, we can also characterize its performance through (15), by approximating perm$[\exp(-\mathbf{W})]$ via a randomized algorithm [16]. In the theorem below, we provide bounds on the expected weight of a ML-match when averaged over all the realizations of the transition time.

THEOREM 1 (EXPECTED ML-WEIGHT). *For the ML rule, the expected weight for a complete batch of footprints, under i.i.d. transition times, satisfies*

$$\mathbb{E}[W^*] \leq \mathbb{E}[N]h(T), \tag{16}$$

where N is the (random) batch size, and $h(T)$ is the differential entropy of the transition time.

Proof: We use the Wald's identity. See [5, A.2]. □

Hence, we see that the average batch size directly influences the bound on the expected weight of the ML-match. The batch sizes depend on a number of factors such the system load, the transition-time pdf and the nature of arrivals, and we will provide numerical results for different scenarios in Section 6.

3.2.2 *Limits of Tracking Performance*

We have so far analyzed the ML-performance for a given transition time pdf. We now analyze the fundamental limits of ML-performance,

$$P_*^{\text{ML}}(\mathbf{Y}_0, \mathbf{Y}_1) := \min_{f_{\mathbf{T}}} P^{\text{ML}}(\mathbf{Y}_0, \mathbf{Y}_1; f_{\mathbf{T}}), \tag{17}$$

where the minimization is over all the joint pdfs $f_{\mathbf{T}}$ (not necessarily with i.i.d. marginals) having their support in the n-dimensional cube $[0, \Delta]^n$. Hence, for a given realization of footprints, finding P_*^{ML} provides a guarantee on the ML-tracking probability P^{ML} under any transition-time pdf satisfying the support bounds. In the theorem below, we prove that P_*^{ML} is at least the reciprocal of the number of unique perfect matchings and also show that the i.i.d. exponential pdf achieves this bound for all batch sizes.

THEOREM 2 (ML GUARANTEE). *Given a batch $(\mathbf{Y}_0, \mathbf{Y}_1)$ of size (n, k) and a class of transition time joint-pdfs with support contained in $[0, \Delta]^n$, and the footprints arrive in the correct order, the ML-guarantee, defined in (17), is*

$$P_*^{ML}(\mathbf{Y}_0, \mathbf{Y}_1) = \frac{1}{N_B(n, k, \mathbf{A})}, \quad n \geq k \geq 0, \tag{18}$$

where $\mathbf{A}$ is the biadjacency matrix constructed in Section 3.1 and $N_B(n, k, \mathbf{A})$ is the number of unique matches in (5). The bound in (18) is achieved only when all the valid matches are equally likely,

1. *The bound in (18) is achieved for all batch sizes when the probability of events $\mathcal{A}_f^{\boldsymbol{\pi}}$, defined in (9), for any valid permutation $\boldsymbol{\pi}$, are of the form*

$$\mathbb{P}[\mathcal{A}_f^{\boldsymbol{\pi}}(\mathbf{Y}_0, \mathbf{Y}_1)] = g[\sum_{i=1}^{k} Y_1(i) + (n-k)Y_1(k) - \sum_{i=1}^{n} Y_0(i)]$$

2. *If we consider only complete batches $(n = k)$, then the bound in (18) is achieved when the joint-pdf of transition time $f_{\mathbf{T}}$ has the form*

$$f_{\mathbf{T}}[T(1), T(2), \ldots, T(n)] = h(\sum_i T(i))$$

3. *When the transition times $T(1), \ldots, T(n)$ are i.i.d., then the bound in (18) is achieved only by the exponential pdf for all batch sizes (and hence, we need to*

allow the deadline $\Delta \to \infty$). On the other hand, if only complete batches are considered, then the bound is also achieved by the uniform distribution $U(0, \Delta)$.

Proof: See [5, A.3]. □

Hence, the performance of the ML-rule for any transition-time pdf is at least the reciprocal of the number of unique perfect matchings in the corresponding bipartite graph. For pdfs achieving this bound, any allowed match is equally likely. Hence, knowledge of the transition-time pdf does not improve performance in this case, and we can use any *distribution-free* matching rule such as the FIFO rule.

Formally, for any transition-time pdf f_T, the performance ratio between ML and FIFO rules (or any other valid perfect matching) satisfies

$$\frac{P^{\text{FIFO}}(n,k,\mathbf{W})}{P^{\text{ML}}(n,k,\mathbf{W})} \geq \gamma, \tag{19}$$

where $\gamma := \frac{\min_{\boldsymbol{\pi}} \mathbb{P}[\mathcal{A}_f^{\boldsymbol{\pi}}(\mathbf{Y}_0,\mathbf{Y}_1)]}{\max_{\boldsymbol{\pi}} \mathbb{P}[\mathcal{A}_f^{\boldsymbol{\pi}}(\mathbf{Y}_0,\mathbf{Y}_1)]}$. Since from Theorem 2, all the valid matches are equally likely for the i.i.d. exponential pdf, $\gamma = 1$, and the FIFO rule has the same performance as any other rule for this case. In the next section, we will compare the ML and FIFO rules for a wider class of pdfs.

3.2.3 FIFO Rule

We now consider a special class of transition time pdf f_T, where all the valid matches are not equally likely, i.e., γ in (19) may not be one, but the ML-rule reduces to the FIFO rule. This implies that the edge weights in (12) satisfy

$$\begin{aligned} W(i_1,j_1) + W(i_2,j_2) &\leq W(i_2,j_1) + W(i_1,j_2), \\ \forall\, i_1 < i_2 \leq k \quad &, \quad j_1 < j_2 \leq k, \end{aligned} \tag{20}$$

and holds for any complete batch of footprints when

$$f_T(t_1) f_T(t_2) \geq f_T(t_1 - \tau) f_T(t_2 + \tau),\ t_1, t_2, \tau \geq 0, \tag{21}$$

and is true for any log-concave pdf, e.g., Rayleigh, lognormal pdfs. When the footprints arrive in the correct temporal order, recall that the ccdf weights are added in (12b), and in this case, we show that the FIFO rule is optimal even for a partial batch in the lemma below.

LEMMA 2 (FIFO IS ML). *If transition-time pdf f_T satisfies (21) and the footprints arrive in the correct order, then FIFO-rule is the optimal ML-rule for all batch sizes.*

Proof: See [5, A.4]. □

Hence, for distributions satisfying (21), optimal tracking with ordered footprints reduces to the FIFO rule that can be performed in linear time. On the other hand, for distributions not satisfying (21), it is not clear if the FIFO-rule performs significantly worse than the optimal rule. We will compare the two through simulations in Section 6.

When the footprints do not arrive in the correct order, the ccdf events are not included in the ML-rule, and only the weights in (12a) are used. Here, the FIFO-rule may not be optimal for a partial batch since it always starts matching with the earliest footprint at S_0. In this case, the optimal rule decides where to start the match based on the edge weights, and runs in $O(n \log k)$ time [2].

4. SEMI-MARKOV PROCESS MODEL

The two-state model studied in the previous section represents a high-level model where the only observable points are the system entry and exit points. When more system points are observable, e.g., the entry and exit points of sub-processes such as authentication, merchandize selection, purchase etc., the two-state model can be expanded to a multi-state model. Assuming that the transition times of each transaction form a multi-state semi-Markov process (SMP) and the transitions of different transactions are independent of one another (see Section 2.1), we consider ML-matching of all the available footprints to the transactions, which generated them.

Since the footprints only carry the timestamps of entry to a state, in general, even the set of states visited by a transaction cannot be tracked with certainty, unlike in a two-state system. Hence, we find the most-likely match between all the available footprints and recall that it is given by the ML-sequence of permutation vectors in (1),

$$[\hat{\boldsymbol{\pi}}_1^{\text{ML}}, \ldots, \hat{\boldsymbol{\pi}}_{N_s}^{\text{ML}}] := \arg\max_{\boldsymbol{\pi}_1,\ldots,\boldsymbol{\pi}_{N_s}} \mathbb{P}(\mathbf{Y}_1^{\boldsymbol{\pi}_1}, \ldots, \mathbf{Y}_{N_s}^{\boldsymbol{\pi}_{N_s}} | \mathbf{Y}_0).$$

By convention, $\boldsymbol{\pi}_0 = \mathbf{I}$. A brute-force search for the ML-sequence of permutation vectors is over all possible footprints paths from the start state S_0 to all the terminating states. It is unclear if this problem has a reduction to bipartite matching, as in the two-state system. However, the search can be simplified through the semi-Markov property which states that the transition time only depends on the current state and the next state, and hence,

$$\mathbb{P}(\mathbf{Y}_1^{\boldsymbol{\pi}_1}, \ldots, \mathbf{Y}_{N_s}^{\boldsymbol{\pi}_{N_s}} | \mathbf{Y}_0) = \prod_{m=1}^{N_s} \mathbb{P}(\mathbf{Y}_m^{\boldsymbol{\pi}_m} | \bigcup_{j \in \mathcal{P}(m)} \mathbf{Y}_j^{\boldsymbol{\pi}_j}). \tag{22}$$

Each term in the product has a structure similar to the two-state system. However, the set of states occurring in any two terms of (22) may not be disjoint, since the sets of immediate predecessors $\mathcal{P}(k)$ and $\mathcal{P}(l)$ of any two states S_k and S_l may not be disjoint. This implies that in general, we cannot independently match the footprints in each term in (22). Hence, we need to and construct high-level states, comprising of many model states that "localize" the movement of footprints, thereby enabling us to perform matching independently within these high-level states. To this end, define a partition of states $(B_m)_{m \geq 0}$ such that states in any two sets in the partition do not share a common immediate predecessor,

$$\mathcal{P}(S_k) \cap \mathcal{P}(S_l) = \emptyset, \quad \forall\, S_k \in B_m, S_l \in B_j, m, j \neq 0, \tag{23}$$

and let $B_0 = S_0$, the start state. Since we have assumed that the state-transition digraph is acyclic (DAG), the partition in (23) is well defined. Therefore, we can rewrite (22) as

$$\mathbb{P}(\mathbf{Y}_1^{\boldsymbol{\pi}_1}, \ldots, \mathbf{Y}_{N_s}^{\boldsymbol{\pi}_{N_s}} | \mathbf{Y}_0) = \prod_{m>0} \mathbb{P}(\bigcup_{S_k \in B_m} \mathbf{Y}_k^{\boldsymbol{\pi}_k} | \bigcup_{S_l \in \mathcal{P}(B_m)} \mathbf{Y}_l^{\boldsymbol{\pi}_l}),$$

where each term in the product corresponds to a bipartite system $(\mathcal{P}(B_m), B_m)$, with the start state $\mathcal{P}(B_m)$ and the terminating state B_m. These bipartite systems are disjoint; a state cannot occur in two systems in the same role, since by definition, B_m are all disjoint sets (partition), and in

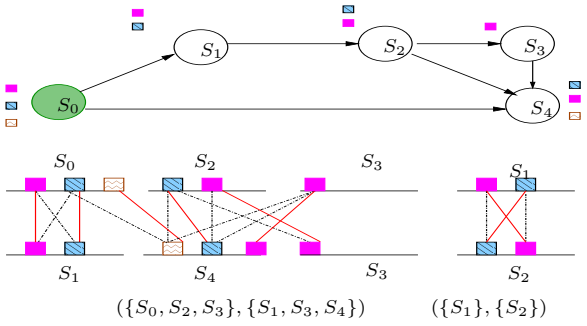


Figure 5: ML-matching of the transactions to their footprints for semi-Markov process model.

Input: SMP with transition DAG G and set of states $\mathcal{S}$
1: $\mathcal{P}$,$\mathcal{N}$ = Imm. predecessor/successor in G
2: Set $B_0 \leftarrow S_0$ (start state), $m \leftarrow 1$
3: **while** $\bigcup_{i=0}^{m-1} B_i \neq \mathcal{S}$ **do**
4: Init $B_m \leftarrow S_i$, for some $S_i \in \mathcal{S} \backslash \bigcup_{i=0}^{m-1} B_i$
5: **repeat**
6: $Prev\_B_m \leftarrow B_m$
7: $B_m \leftarrow \mathcal{N}(\mathcal{P}(B_m))$
8: **until** $B_m == Prev\_B_m$
9: $m \leftarrow m+1$
10: **end while**, **return** $(B_m)_{m\geq 0}$

Figure 6: Partitions of states satisfying (23).

(23), we also require the sets $\mathcal{P}(B_m)$ to be disjoint. Hence, we can conduct decentralizing matching in these bipartite systems. This also implies that knowledge of the footprints and the model parameters (such as the transition-time pdf) is only required "locally" within each bipartite system. After undertaking matching in all the bipartite systems, the most-likely sequence of footprints produced by each transaction are constructed by splicing together the results of matching in bipartite systems to obtain the set of permutation vectors $\boldsymbol{\pi}_i^{\mathrm{ML}}$ in (1). In the lemma below, we provide some properties of the partition (B_m).

LEMMA 3 (PROPERTIES OF PARTITION $(B_m)_{m\geq 0}$). *For a semi-Markov process with acyclic transition digraph and a partition of states $(B_m)_{m\geq 0}$, the following properties hold for any finite number of recursions iff (23) is true:*

$$\mathcal{N}(\mathcal{P}\ldots(\mathcal{N}(\mathcal{P}(B_m)))) \subset B_m, \quad m>0, \tag{24}$$
$$\mathcal{P}(\mathcal{N}\ldots(\mathcal{P}(B_m))) \subset \mathcal{P}(B_m), \quad m>0, \tag{25}$$

Proof: See [5, A.5]. □

From the above lemma, all the immediate successors of any state in $\mathcal{P}(B_m)$ are in B_m and similarly all the immediate predecessors of any state in B_m are in $\mathcal{P}(B_m)$. Hence, the set of states in $(\mathcal{P}(B_m), B_m)$ represents complete one-step forward and backward reachability. We use the properties (24) and (25) in Lemma 3 to generate the unique partition (B_m) in Fig.6, where the algorithm terminates in finite time due to the acyclicity of the state transition digraph and has to be run only once prior to monitoring.

We now specify the nodes and the edge weights for each bipartite system $(\mathcal{P}(B_m), B_m)$. Along the lines of (3), we define $\mathrm{Cnt}(\mathcal{P}(B_m))$ as the number of instances residing at $\mathcal{P}(B_m)$ at the time of observation, given by

$$\mathrm{Cnt}(\mathcal{P}(B_m)) = |\mathbf{Y}_{\mathcal{P}(B_m)}| - |\mathbf{Y}_{B_m}|. \tag{26}$$

A batch of footprints is defined between the successive zero-crossings of $\mathrm{Cnt}(\mathcal{P}(B_m))$. Along the lines of (12), given the transition probability matrix $\mathbf{P}$ of the SMP, for any states $S_k \in B_m, S_l \in \mathcal{P}(B_m)$ and i.i.d. transition times drawn from pdf $f_{T_{k,l}}$, the edge weight between the i^{th} footprint at S_k and the j^{th} footprint at S_l of a batch is given by

$$W(i,j;S_k,S_l) := -\log[P(k,l) f_{T_{k,l}}(Y_l(j) - Y_k(i))],$$
$$\forall\, S_k \in \mathcal{P}(S_l), \quad 1 \leq i,j \leq |\mathbf{Y}_k|, 1 \leq j \leq |\mathbf{Y}_l|. \tag{27}$$

For a partial batch, we have $\mathrm{Cnt}(\mathcal{P}(B_m)) > 0$ and some instances are still residing at $\mathcal{P}(B_m)$. When the footprints arrive in the correct temporal order, we define the ccdf events along the lines of (9) for each state $S_k \in \mathcal{P}(B_m)$, whenever there are more footprints at S_k than the total at all its immediate successors, i.e., $\mathrm{Cnt}(S_k) = |\mathbf{Y}_k| - \sum_{l\in\mathcal{N}(k)} |\mathbf{Y}_l| > 0$. The probability that transaction corresponding to the i^{th} footprint at S_k is still resident at S_k is given by

$$P_{\mathrm{ccdf}}(i,S_k) := \sum_{l\in\mathcal{N}(k)} P(k,l)\bar{F}_{T_{kl}}[Y_l(|\mathbf{Y}_l|) - Y_k(i)],$$

where $Y_l(|\mathbf{Y}_l|)$ is the most recent footprint at state S_l. In this case, the ccdf edge-weights corresponding to state S_k are

$$W(i,\delta_k;S_k) := -\log P_{\mathrm{ccdf}}(i,S_k), \quad \forall\, 1 \leq i \leq |\mathbf{Y}_k|, \tag{28}$$

and as in Section 3.1, $\mathrm{Cnt}(S_k)$ number of identical copies of the ccdf node δ_k are added to the bipartite graph. We now state the result that the optimal ML-rule maximizing the probability of all the footprints being matched correctly to the instances generating them is given by decentralized minimum-weight matching in each bipartite system.

THEOREM 3 (ML-RULE IN SMP). *Given a semi-Markov process with an acyclic state transition digraph and i.i.d. transaction transitions, the ML-rule is given by the decentralized minimum-weight matching in the bipartite systems $(\mathcal{P}(B_m), B_m)$, where partition (B_m) is given by the algorithm in Fig.6 and the edge weights for each bipartite system are given by (27) and (28). The probability that all the transactions are correctly tracked correctly is the product of the ML-probabilities of all the bipartite systems, given by (15).*

Proof: From the semi-Markov property and Lemma 3. □

An example of a SMP is shown in Fig.5 with partition $B_0 = \{S_0\}$, $B_1 = \{S_1, S_3, S_4\}$, $B_2 = \{S_2\}$ and two bipartite systems $(\{S_0, S_2, S_3\}, \{S_1, S_3, S_4\})$ and $(\{S_1\},\{S_2\})$. Although the state S_3 occurs in both the high-level states $\mathcal{P}(B_1)$ and B_1, the edges in the bipartite graph are constructed only for valid paths in the transition digraph. Matching is done jointly among the states in a bipartite system, but independently across systems and batches, and the results are spliced together.

Hence, from Theorem 3, the ML-rule can be decomposed into decentralized matching in the bipartite systems and,

hence, its complexity is greatly influenced by the number of such bipartite systems; if the number is large, then the transactions at different states do not interleave to a large extent. However, in the worst case, there may be only a single partition $B_1 = \mathcal{S} \backslash S_0$, containing all the states except S_0, e.g., when all the states are immediate successors of S_0. On the other hand, for a fixed number of states $N_s + 1$, the best-case scenario is the linear transition digraph, where there are N_s number of bipartite systems. The linear graph is a special case of the tree and the number of bipartite systems in a tree is equal to the number of non-terminating states in the tree and each bipartite system consists of a non-terminating state and its immediate successors (children). We utilize the property that any connected acyclic digraph can be expanded to a tree to derive bounds on the number of sets in the partition for any acyclic digraph in [5, A.6]. We also use the properties of the partition to find other performance measures, such as bounds on the total number of transactions resident at a state, in [28].

5. PRESENCE OF TOKENS

We have so far considered the worst-case scenario where there are no transaction identifiers or tokens in any of the footprints. However, in a real system, while certain footprints may lack tokens, others may include them, e.g., newly developed systems built using ARM instrumentation [1] to generate tokens may be combined with legacy systems lacking such instrumentation. In this section, we consider the case when footprints at a certain subset of the model states carry tokens. We provide the optimal ML-rule for the special cases of linear and tree state-transition digraphs.

We first consider the SMP with a linear state-transition digraph and assume that the footprints have tokens both at the start state S_0 and the terminating state S_{N_s}. An example is shown in the top portion of Fig.7. This scenario can be easily extended to the case where tokens are available at one or more intermediate states of the linear SMP, since matching can be broken down to multiple parts. It is not clear if the ML rule has a decentralized implementation here, as in the tokenless case in Theorem 3. In the theorem below, we provide the decentralized ML-rule.

We first define some new notations. For a linear SMP with N_s+1 number of states, the composition of ML-permutation vectors between states S_a and S_b, for $0 \le a < b \le N_s$,

$$\hat{\Pi}^{\mathrm{ML}}_{a,b}(j) := \pi_{c+1}(\pi_c \dots \pi_{a+1}(j)),\ 1 \le j \le |\boldsymbol{\pi}_a|, \qquad (29)$$

where π_k is the ML-permutation (minimum weight match) for the bipartite system (S_k, S_{k+1}), with $c = \mathrm{end}(j; a, b)$, defined as

$$\mathrm{end}(j; a, b) := \arg \min_{a \le c \le b} I(\hat{\pi}^{\mathrm{ML}}_{c+1}(\hat{\pi}^{\mathrm{ML}}_c \dots \hat{\pi}^{\mathrm{ML}}_a(j)) = \delta_c),$$

where δ_c is the ccdf node representing the event that the transaction that generated the j^{th} footprint at S_a is currently residing at state S_c. Define the events

$$A(i) = [\text{token of } i^{\text{th}} \text{ transaction is seen at } S_{N_s}], \qquad (30)$$
$$B(j) := [\mathrm{end}(j; 1, N_s - 1) == N_s - 1]. \qquad (31)$$

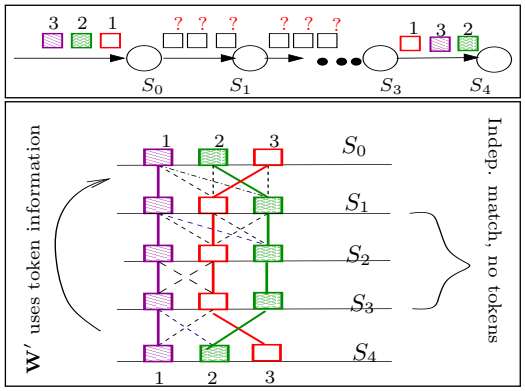


Figure 7: ML-rule for linear transition digraph with tokens at S_0 and S_4. See Theorem 4.

Hence, $B(j)$ in (31) is the event that the intermediate ML-path starting with the j^{th} footprint at S_1 ends with an actual footprint at S_{N_s-1} and not a ccdf node. Since tokens are produced at S_{N_s}, when $I_{A(i)} = 1$ (I is indicator function), the permutation $\pi_{N_s}(i) = g(i)$, where g is a fixed function providing the footprint index at S_{N_s} with the same token as i^{th} transaction (footprint at S_0).

THEOREM 4 (ML-RULE WITH TOKENS). *For a linear semi Markov process with tokenized start state S_0 and terminating state S_{N_s}, the ML-rule is to*

1. *find minimum weight matching in bipartite systems (S_{k-1}, S_k), $2 \le k \le N_s - 1$, with weights $\mathbf{W}_k$ in (12),*
2. *obtain the intermediate ML-paths through $\hat{\mathbf{\Pi}}^{ML}_{2,N_s-2}$, the composition of the ML-permutations defined in (29),*
3. *finally, match jointly the tokenized footprints at states S_0 and S_{N_s} to the intermediate-ML paths, obtained in step 2, using a new weight matrix $\mathbf{W}' = [W'(i,j)]$, where $W'(i,j)$ is the edge weight between the i^{th} transaction and the intermediate ML-path that originates with the j^{th} footprint at state S_1,*

$$W'(i,j) = W_1(i,j) + \begin{cases} 0, & \text{(32a)} \\ W_{N_s}(\hat{\Pi}^{ML}_{2,N_s-1}(j), \delta_{N_s-1}), & \text{(32b)} \\ W_{N_s}(\hat{\Pi}^{ML}_{2,N_s-1}(j), g(i)), & \text{(32c)} \\ \infty, & \text{(32d)} \end{cases}$$

where, for events $A(i)$, $B(j)$ defined in (30), (31),

$$\begin{aligned} \textit{Case a} &: I_{A(i)} = 0, I_{B(j)} = 0, \\ \textit{Case b} &: I_{A(i)} = 0, I_{B(j)} = 1, \\ \textit{Case c} &: I_{A(i)} = 1, I_{B(j)} = 1, \\ \textit{Case d} &: I_{A(i)} = 1, I_{B(j)} = 0. \end{aligned}$$

Proof: See [5, A.7]. □

Hence, for a linear SMP, even in the presence of tokens, the ML-rule can be implemented in a decentralized manner at all the tokenless states to obtain the intermediate

ML-paths. The only additional complexity is in combining the footprint information at the tokenized states and then matching them jointly to the intermediate ML-paths using the modified weight matrix $\mathbf{W}'$ in (32). In Fig.7, we depict the ML-matching in a five state linear SMP for a complete batch of three footprints. The ML-match between the footprints at states S_1 and S_2 are conducted independent of the other footprints, similarly for the match between the footprints at states S_2 and S_3. The matching results are then spliced together to form intermediate ML-paths from S_1 to S_3, which are then matched to the tokenized footprints at S_0 and S_{N_s} using the weights in (32).

For a complete batch of transactions, all the transactions have completed their operations producing all the tokenized footprints at S_{N_s} and there are no ccdf nodes, implying that (32) reduces to (32c). However, for a partial batch with footprints arriving in order, any of the four cases in (32) can occur. For case a, since the tokenized footprint for i^{th} instance is not yet seen at S_{N_s} ($I_{A(i)} = 0$) and the j^{th} intermediate ML-path already has a ccdf node as its endpoint ($I_{B(j)} = 0$), the original (tokenless) edge weight $W_1(i,j)$ is unchanged. For case b, $I_{B(j)} = 1$ and hence, the intermediate ML-path needs to be matched to both a starting point (footprint at S_0) and an end point; the endpoint has to be the ccdf node δ_{N_s-1} since the i^{th} instance is most-likely still residing at S_{N_s-1} ($I_{A(i)} = 0$). For case c, the tokenized footprint at S_{N_s} is available ($I_{A(i)} = 1$) and its information is combined with the corresponding footprint at S_0. For this case, only the intermediate ML-paths that need both a start and an end point ($I_{B(j)} = 1$) are eligible candidates to be matched to the i^{th} transaction and hence, we do not allow the match when $I_{B(j)} = 0$, by having an infinite edge weight (case d).

We now consider extending the ML-rule to the case where the state-transition digraph is a tree and only some of the terminating states are tokenized. Even in this case, we can conduct decentralized minimum weight matching in the bipartite systems, consisting of only tokenless states, to obtain intermediate ML-paths. However, in contrast to Theorem 4, the matching the intermediate ML-paths to the remaining footprints has additional complexity since not all the remaining footprints are tokenized. In fact, this problem of ML-matching in a tokenized tree reduces to a *transportation problem with exclusionary side constraints* [15], an NP-hard problem, and not to a bipartite matching. The transportation problem consists of sources and destinations, with each destination requiring a certain set of sources and vice-versa, and a cost is associated with each source-destination pair. The exclusionary side constraint additionally requires that certain groups of sources not be assigned to the same destination. In our problem, the sources are the candidate start points (tokenized footprints at S_0) and the end points of a footprint path (tokenized and tokenless footprints at the terminating states in the tree) and the destination nodes are the intermediate ML-paths. Each destination node requires two source nodes and we have the exclusionary side constraints, that any two source nodes of the same type (start or end) cannot be assigned to the same intermediate ML-path, and that each tokenized footprint at a terminating state has to assigned to the same destination as the corresponding footprint at S_0. In general, the above reduction does not hold for a general SMP, which is not a tree, since such a construction of intermediate ML-paths may not be feasible, e.g., for the SMP in Fig.5, when S_4 is tokenized, intermediate path, consisting of footprints at states S_1, S_2, and S_3 cannot be constructed.

6. PERFORMANCE RESULTS

In this section, we evaluate the performance of tracking rules and the tightness of the bounds on the matching probabilities, derived in Sections 3, 4 and 5. The results here are primarily for the two-state model due to its central role in our theoretical analysis. We also consider two cases of the multi-state SMP model, viz., the linear SMP and the SMP shown in Fig.5. We study the effect of increasing the system load on the tracking performance.

6.1 Setup

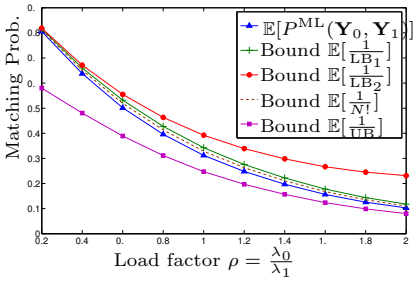
We consider a two-state system with start state S_0 and end state S_1. We assume that new transactions enter the system according to a Poisson process with rate λ_0, i.e., the transaction inter-arrival times are exponentially distributed with mean λ_0^{-1}. We assume that the transition times are independent of the system load, as we did in our analysis in Sections 3, 4 and 5. We consider the exponential, uniform and heavy-tailed pdfs, the Pareto and the log-normal pdfs, for the transition times. We compare the performance of different pdfs by fixing the mean transition time, denoted by λ_1^{-1}. We vary the degree of interleaving and thereby, the batch size and the matching performance, through the mean transition time λ_1^{-1}. We study even the extreme (atypical) operational conditions with large degree of interleaving, where the *load factor* $\rho = \frac{\lambda_0}{\lambda_1}$, may be greater than one.

We conduct real-time matching for 100 transaction instances entering the system and average the values over 100 independent simulation runs. Recall that we divide the footprints into batches in real time, as described in Section 3.1. We use the simulation runs to compute the empirical fraction of batches, either complete or partial, with all the footprints matched correctly and this approximates the expected matching probability $\mathbb{E}[P^{\text{ML}}]$ in (15).

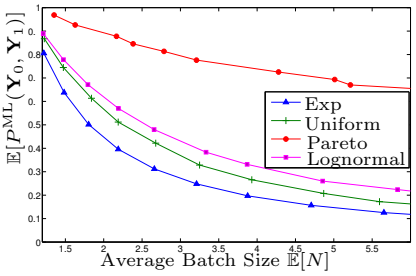
6.2 Results

In Fig.8a, we plot the matching probability $\mathbb{E}[P^{\text{ML}}]$ and its bounds for the exponential pdf, where P^{ML} for a given pdf is given in (15) and reduces to (18) for the exponential pdf. We find that the bounds for a general bipartite graph, given by LB_1 and $n!$ in [5, (33)], closely approximate $\mathbb{E}[P^{\text{ML}}]$. On the other hand, the elementary-graph bounds LB_2 and UB_2 in [5, (34)] are relatively loose. The lower and the upper bounds in [5, (33)] are tight for dense graphs, while the elementary-graph bounds in [5, (34)] are tight for sparse graphs. This implies that the batches formed here are densely connected. In Fig.8b, we plot $\mathbb{E}[P^{\text{ML}}]$ against the average batch size $\mathbb{E}[N]$ and find that the exponential distribution has the minimum performance. This is also seen in Fig.9c, for the case of the multi-state SMP in Fig.5.

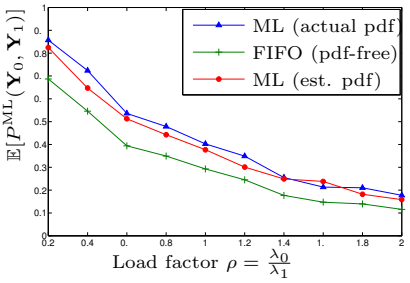
In Fig.8c, we compare the performance of the optimal ML-rule with the distribution-free first-in-first-out (FIFO) rule for the Pareto pdf. From Lemma 2, the FIFO rule is optimal for other distributions considered here, since they satisfy (21). We also consider the case when the ML-rule is employed on the pdf, estimated using a separate set of 100 learning samples through kernel-smoothing methods. We find that there is some performance gap between the ML and the FIFO rules. On the other hand, there is hardly any performance loss due to pdf estimation. In Fig.9a, for



(a) Matching probability and its bounds for exponential pdf.

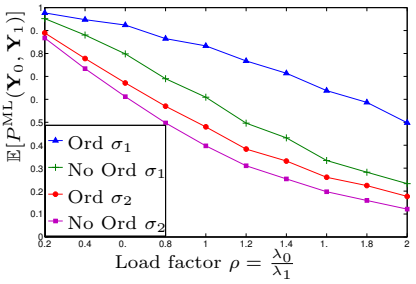


(b) Matching probability vs. average batch size.

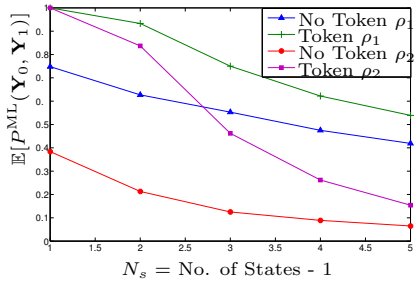


(c) ML and FIFO rule for Pareto pdf with shape parameter $K = 1.15$.

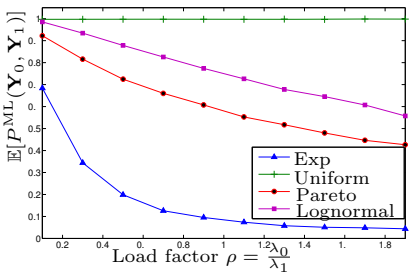
Figure 8: Simulation results to show tightness of bounds and matching probabilities for different distributions.



(a) Effect of order under ML-rule for log-normal pdf. $\sigma_1 = 0.1, \sigma_2 = 0.5$.



(b) Linear SMP with exponential pdf. Load factors $\rho_1 = 0.1, \rho_2 = 0.4$



(c) Matching probabilities for the semi-Markov process in Fig.5.

Figure 9: Simulation results to show the effect of arrival of footprints in correct order and token availability.

the log-normal pdf, we study the cases when the footprints arrive/do not arrive in the correct order. We find that the presence of a correct time order improves matching performance for partial batches.

In Fig.9b, we consider complete batches in a linear SMP with $N_s + 1$ number of states. We plot the matching probability $\mathbb{E}[P^{\mathrm{ML}}]$ against N_s and compare the cases when tokens are absent/present at the terminating state S_{N_s}, and use the rules in Theorems 3 and 4. We find that although tokens improve performance, their efficacy reduces as we increase the number of states or the load factor.

6.3 Implications

Our experiments confirm that the exponential pdf serves as a benchmark and has the worst-case tracking performance for a wide variety of systems and conditions. We find that the probability that all the transactions are matched correctly decreases rapidly on increasing the number of transactions and model states. Even the presence of tokens is not helpful in improving the matching probability if there is a large separation between the tokenized states or high load factor. We also find that the performance gap between the optimal ML-matching rule and the distribution-free FIFO rule can be quite large for pdfs not obeying (21). On the other hand, for such pdfs, there is hardly any performance loss if we estimate the pdf using learning samples. Hence, rather than use the FIFO rule, it is better to estimate the pdf, whenever possible. We also find that when footprints arrive in the correct temporal order at the monitoring engine, there is significant improvement in real-time matching performance.

7. RELATED WORK

The classical work on monitoring by Chandy and Lamport considers the detection of certain global system states [10], and this work started a flurry of research activity, outlined in the survey [21]. However, these studies do not deal with the issue of tracking individual transaction instances, which is the focus of this paper. Industry standards like the open-group ARM instrumentation [1] provide libraries for instrumenting applications and generating transaction correlators or tokens that may be used to track the flow of control across different transaction segments. "Whitebox" methods, presented in [11, 12, 27], use such instrumentation-based tokens for fine-grained tracking. However, such intrusive techniques are often not applicable in enterprise environments. Existing "blackbox" methods, presented in [4, 19], avoid such instrumentation. However, they either do not track individual call flow or use app-specific event-schema [6]. Various statistical techniques have been employed for monitoring. In [22], Mendes and Reed consider monitoring of large distributed systems with large amounts of data through statistical sampling. Distributed monitoring of petri-net based concurrent systems is considered in [7, 17, 24].

In contrast to the previous works, this paper assumes no instrumentation, correlators or infrastructure specifics and provides the optimal rule for tracking individual trans-

actions in a non-intrusive manner. In a related publication [28], we additionally describe our monitoring-system architecture, provide experimental results on the response time and also characterize bounds on the aggregate number of transaction instances present at different states at any time instant. Such aggregate information can be used for dynamic resource allocation and load balancing strategies.

We have assumed that the transaction model is perfectly known and may be obtained from knowledgeable users, e.g., system administrators. Our work does not deal with the issues of discovering the model, a challenging problem in itself. Agrawal, Gunopulos, and Leymann introduced the idea of process mining in the context of workflow management in [3], and since then, several process-mining techniques have been proposed and analyzed [13, 20, 25, 26, 29, 31].

8. CONCLUSION

The end-to-end tracking of transaction instances is of great importance in the enterprise environments. However, many factors significantly reduce the capability to track transactions, e.g., presence of legacy systems, impossibility of accessing the underlying code, and so on. Hence, it is critical to explore the feasibility of tracking transactions using only time-stamped footprints, something that can be safely assumed to exist in most application logs. The distinguishing feature of the current effort from the previous "blackbox" type of studies is the absence of any assumptions on the presence of tokens in the observed footprints. This paper is the first study to formally analyze the optimal tracking rules and the resulting performance for a wide range of system models.

We acknowledge the various limitations of this paper. We have only analyzed, both theoretically and experimentally, the probability that all the instances are matched correctly, a stringent condition. Alternatively, we can use the (approximate) marginal probability as the performance metric for each transaction under the bottleneck formulation [9]. We have assumed that the transactions progress according to a semi-Markov process model; this model cannot represent concurrency within a transaction instance and petri-net models are needed to handle such systems. Our assumptions of a perfectly known transaction model and transition-time pdfs need to be relaxed. We have also not considered the design of dynamic matching algorithms that enhance real-time transaction tracking. These are some challenging directions to explore.

Acknowledgements

The authors thank the anonymous reviewers and B. Urgaonkar for comments which substantially improved this paper. The authors also thank N. Banerjee and B. Sengupta for fruitful discussions and for sharing their log-generation code. The second and third authors thank P. Hurley for collaboration on broader aspects of this work; P. Klein, S. Perelman and D. Verma for their support. The first author thanks her advisor Prof. L. Tong for his continued support.

9. REFERENCES

[1] ARM. `http://www.opengroup.org/tech/management/arm/`.

[2] A. Aggarwal, A. Bar-Noy, S. Khuller, D. Kravets, and B. Schieber. Efficient minimum cost matching using quadrangle inequality. In *Proc. of FOCS*, pages 583–592, 1992.

[3] R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Proc. of Intl. Conf. on Extending Database Technology*, pages 469–483, 1998.

[4] M. Aguilera, J. Mogul, J. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *Proc. of SOSP*, pages 74–89, 2003.

[5] A. Anandkumar, C. Bisdikian, and D. Agrawal. Tracking in a Spaghetti Bowl: Monitoring Transactions Using Footprints. Technical Report RC24422 (W0711-107), IBM Research, Yorktown Heights, NY 10598, November 2007. `http://domino.watson.ibm.com/library/CyberDig.nsf/home`.

[6] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modelling. In *Symp. on Operating Sys. Design & Implementation*, 2004.

[7] A. Benveniste, S. Haar, E. Fabre, and C. Jard. Distributed monitoring of concurrent and asynchronous systems. In *ATPN-Workshop on Discrete Event Sys. Control*, 2004.

[8] A. Borr. Transaction monitoring in Encompass [TM]: Reliable distributed transaction processing. In *Proc. of VLDB*, 1981.

[9] R. Burkard. Selected topics on assignment problems. *Discrete Applied Mathematics*, 123(1-3):257–302, 2002.

[10] K. Chandy and L. Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Tran. on Computer Systems*, 3(1):63–75, 1985.

[11] M. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer. Path-based failure and evolution management. In *Symposium on Networked System Design and Implementation*, 2004.

[12] M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: problem determination in large, dynamic Internet services. *Dependable Sys. & Networks*, pages 595–604, 2002.

[13] J. Cook and A. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Tran. on Software Engineering and Methodology*, 7(3):215–249, 1998.

[14] R. Gallagher. *Discrete Stochastic Processes*. Springer, 1996.

[15] D. Goossens and F. Spieksma. The Transportation Problem with Exclusionary Side Constraints. *4OR*, 2007.

[16] M. Jerrum, A. Sinclair, and E. Vigoda. A Polynomial-Time Approximation Algorithm for the Permanent of a Matrix with Nonnegative Entries. *Journal of ACM*, 51(4):671–697, 2004.

[17] G. Lamperti and M. Zanella. *Diagnosis of Active Systems: Principles and Techniques*. Kluwer Academic Publishers, 2003.

[18] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover Publications, 2001.

[19] H. Liu, H. Zhang, R. Izmailov, G. Jiang, and X. Meng. Real-time Application Monitoring and Diagnosis for Service Hosting Platforms of Black Boxes. In *IEEE Intl. Symposium on Integrated Network Management*, pages 216–225, 2007.

[20] K. Magoutis, M. Devarakonda, and K. Muniswamy-Reddy. Galapagos: Automatically Discovering Application-Data Relationships in Networked Systems. In *IEEE Intl. Symposium Integrated Network Management*, pages 701–704, 2007.

[21] M. Mansouri-Samani and M. Sloman. Monitoring distributed systems. *Network, IEEE*, 7(6):20–30, 1993.

[22] C. Mendes and D. Reed. Monitoring Large Systems Via Statistical Sampling. *Intl. Journal of High Performance Computing Applications*, 18(2):267, 2004.

[23] P. Ostrand. Systems of distinct representatives, II. *J. Math. Anal. Appl*, 32:1–4, 1970.

[24] Y. Pencole, M. Cordier, L. Roze, and R. Irisa. A decentralized model-based diagnostic tool for complex systems. In *Proc. on Tools with Artificial Intelligence*, pages 95–102, 2001.

[25] W. Peng, T. Li, and S. Ma. Mining logs files for data-driven system management. *SIGKDD Explorations*, 7(1):44–51, 2005.

[26] J. Rouillard. Real-time Log File Analysis Using the Simple Event Correlator (SEC). In *USENIX System Administration Conf.(LISA)*, pages 133–149, 2004.

[27] M. Schmid, M. Thoss, T. Termin, and R. Kroeger. A Generic Application-Oriented Performance Instrumentation for Multi-Tier Environments. In *IEEE Intl. Symposium on Integrated Network Management*, pages 304–313, 2007.

[28] B. Sengupta, N. Banerjee, A. Anandkumar, and C. Bisdikian. Non-Intrusive Transaction Monitoring Using System Logs. In *Proc. of IEEE/IFIP Network Operations & Management Symposium (NOMS)*, Salvador, Brazil, April 2008.

[29] R. Vaarandi and T. Tehnikaülikool. *Tools and Techniques for Event Log Analysis*. Tallinn Univ. of Technology Press, 2005.

[30] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[31] W. van der Aalst and B. van Dongen. Discovering Workflow Performance Models from Timed Logs. In *Proc. of EDCIS*, volume 2480, pages 45–63, 2002.