

Stochastic Petri Net Analysis of a Replicated File System

Joanne Bechta Dugan, Gianfranco Ciardo
Department of Computer Science
Duke University
Durham, NC 27706, USA

September 3, 2002

Abstract

We present a stochastic Petri net model of a replicated file system in a distributed environment where replicated files reside on different hosts and a voting algorithm is used to maintain consistency. Witnesses, which simply record the status of the file but contain no data, may be used in addition to or in place of files to reduce overhead. We present a model sufficiently detailed to include file status (current or out-of-date) as well as failure and repair of hosts where copies or witnesses reside. The number of copies and witnesses is not fixed, but is a parameter of the model. Two different majority protocols are examined, one where a majority of all copies and witnesses is necessary to form a quorum, the other where only a majority of the copies and witnesses on operational hosts is needed. The latter, known as adaptive voting, is shown to increase file availability in most cases. We also investigate the process of selection of copies and witnesses to participate in an update when more than the majority is available and show the inherent performance/reliability tradeoffs.

1 Introduction

Users of distributed systems often replicate important files on different hosts, to protect them from a subset of host failures. The consistency of the files across the database is difficult to maintain manually, and so a data abstraction, the *replicated file*, has been introduced to automate the update procedure [Ellis83, Popek81, Stonebraker79]. The consistency of the files is most often maintained by assigning *votes* to each copy of the file and by automatically assembling a quorum (majority) of votes for a file access. Requiring a majority for each update insures that at most one write set can exist at any time, and that a quorum automatically includes at least one of the most recently updated copies. Such a system tolerates host failures to the extent that a minority of votes may be unavailable at any time, but a file update will still be permitted.

Pâris suggests replacing some copies with *witnesses*, which contain no data but which can testify to the current state of the copies. Witnesses have low storage costs and are simple to update [Paris86a, Paris86]. In [Paris86] it is believed that the replacement of some copies with witnesses has a minor impact on the availability of the file system.

When an availability analysis of such a replicated file system is performed, a Markov chain is often used [Jajodia87, Paris86a, Paris86]. If the Markov chain is constructed manually, the analysis is often limited to small systems (perhaps three copies) modeled at the highest level. Without the

proper tools, it is difficult to develop a Markov model of a large system detailed enough to include both failure and repair of hosts, and voting and updating procedures. A higher level “language” is needed for the description of the system to allow the automatic generation of the Markov chain. The stochastic Petri net model provides such a language. We present a stochastic Petri net model for a distributed file system, whose structure is independent of the number of copies and witnesses. This model is automatically converted into a Markov chain and solved numerically for the state probabilities (the generated Markov chains have up to 1746 states).

After describing the replicated file system being considered and defining the stochastic Petri net model, we incrementally develop a model that includes copies and witnesses, failure and repair, requests and voting. We then solve the model for the availability of the filesystem under static and adaptive voting algorithms. Then we examine the performance/reliability tradeoffs associated with preferring copies *vs.* preferring witnesses to participate in a quorum, when more than a majority are available.

2 Replicated File System

Mutual consistency of the various representatives of a replicated file can be maintained in at least three ways [Ellis83]: writing to all copies at each update, designating a ‘primary’ copy as leader [Parker83, Stonebraker79], or using various weighted voting algorithms [Davcev85, Ellis83, Gifford79, Jajodia87, Paris86a, Paris86, Thomas79]. In a weighted voting algorithm, each copy is assigned a number of votes; a set of copies representing a majority of votes (a quorum) must be assembled for each update.

Copies may be assigned different weights (number of votes), including none, and different quorums can be defined for read and write operations. Consistency is guaranteed as long as the quorums are high enough to disallow simultaneous read and write operations on disjoint sets of copies. The simplest quorum policies require a majority of copies to participate in any read or write, by assigning one vote to each. However, the system administrator can alter the performance by manipulating the voting structure of a replicated file [Gifford79].

Associated with each representative is a timestamp, or version number, which is increased each on each update. At any time, the copies representing a majority of votes have the same (highest) version number and the assemblage of a majority is guaranteed to contain a representative with the highest version number. Copies with the same version number are identical. Each time a quorum of copies is gathered, any out-of-date copy participating in the quorum is brought up-to-date before the transaction is processed. Copies not participating in the quorum become obsolete, since they will no longer have the highest version number.

Pâris suggests replacing some of the copies by *witnesses* to decrease the overhead of maintaining multiple copies of a file. Witnesses contain no data, but can vote to confirm the current state of copies. They can be created and maintained easily; they simply contain version numbers reflecting the most recent write observed. A quorum can be gathered counting witnesses as copies, but at least one current copy must be included. If there is no copy with the same version number as the highest witness’ version number, then an update cannot be performed. Pâris further suggests transforming witnesses into copies and vice versa, as needed.

3 Assumptions

In the model developed, we assume for simplicity that each representative (copy or witness) is assigned one vote. This vote assignment is called the *uniform assignment* in [Barbara87], where it is shown to optimize reliability for fully connected, homogeneous systems with perfect links. We also assume that copies and witnesses are not transformed into the other. Further, we assume the following scenario for the gathering of a quorum. When a request is received, a status request message is sent to each site known to contain a representative. Each representative residing on a functioning host computer answers, communicating its status. If more than a majority of representatives are available, a subset is chosen to participate in the update. The selection process of the members in the participating subset tries to minimize the time needed to service the request by including as many current representatives as possible, preferably witnesses since they are fast to update. A quorum is thus formed by first choosing a current copy, then, as needed, current witnesses, more current copies, outdated witnesses, and outdated copies, in that order. All current copies and witnesses participating in the quorum are updated, while the remaining ones become outdated, since they will not have the most recent timestamp.

The host computers can fail and be repaired, thus not all representatives are available at all times. A representative on a machine that has failed is assumed to be out-of-date when the machine is subsequently repaired. This is a conservative, but not unrealistic, assumption. It is possible to exactly model whether a representative has gone out-of-date while the host was down, but we chose not to do so because the model would be more complex. If a quorum cannot be formed when a request is received, a manual reconstruction is initiated, to restore the system to its original state.

Again, to keep the model from becoming too complex, we only consider write requests and we assume that there is a time lapse between requests. We also assume that hosts do not fail while a quorum is being gathered. Hosts may fail during the update procedure. All times between events are assumed exponentially distributed, to allow a Markov chain analysis. If the times have a general distribution, semi-Markov analysis or simulation would be needed [Bechta84a].

4 Stochastic Petri Net

A Petri net is a graphical model useful for modeling systems exhibiting concurrent, asynchronous or nondeterministic behavior [Peterson81]. The nodes of a Petri net are *places* (drawn as circles), representing conditions, and *transitions* (drawn as bars), representing events. *Tokens* (drawn as small filled circles) are moved from place to place when the transitions *fire*, and are used to denote the conditions holding at any given time. As an event is usually enabled by a combination of conditions, a transition is enabled by a combination of tokens in places. An arc is drawn from a place to a transition or from a transition to a place. Arcs are used to signify which combination of conditions must hold for the event to occur and which combination of conditions holds after the event occurs. If there is an arc from place p to transition t , p is an *input place* for t , if there is an arc from t to p , p is an *output place* for t . (These conditions are not exclusive.)

A transition is *enabled* if each input place contains at least one token; an enabled transition fires by removing a token from each input place and depositing a token in each output place.

Stochastic Petri nets (SPN) were defined by associating an exponentially distributed *firing time* with each transition [Molloy81, Natkin80]. A SPN can be analyzed by considering all possible *markings* (enumerations of the tokens in each place) and solving the resulting *reachability graph* as a Markov chain. Generalized stochastic Petri nets (GSPN) allow immediate (zero firing time)

and timed (exponentially distributed firing time) transitions; immediate transitions are drawn as thin bars, timed transitions are drawn as thick bars. GSPN are solved as Markov chains as well [Ajmone84]. Extended stochastic Petri nets (ESPN) [Bechta84a, Bechta85] allow transition times to be generally distributed. In some cases an ESPN can be solved as a Markov chain or as a semi-Markov process, otherwise it can be simulated. The model used in this paper is basically the GSPN with variations from the original (or the current) definition, so we will use the term stochastic Petri net (SPN) with the generic meaning of “Petri net with stochastic timing”.

Three additional features to control the enabling of transition are included in our SPN model: *inhibitor arcs*, *transition priorities*, and *enabling functions*. An inhibitor arc [Peterson81] from a place to a transition disables the transition if the corresponding input place is not empty. If several transitions with different priorities are simultaneously enabled in a marking, only the ones with the highest priority are chosen to fire, while the others are disabled. An enabling function is a logical function defined on the marking, if it evaluates to *false*, it disables the transition. More specifically, a transition t is enabled if and only if (1) there is at least one token in each of its input places, (2) there is no token in any of its inhibiting places, (3) its enabling function evaluates to *true*, and (4) no other transition u with priority over t and satisfying (1), (2), and (3) exists. In the description, each transition is labeled with a tuple (“name”, “priority”, “enabling function”), where the enabling function is identically equal to *true* if absent. The priority is indicated by an integer; a lower number indicates a higher priority.

If several enabled transitions are scheduled to fire at the same instant, a probability distribution (possibly marking-dependent) is defined across them to determine which one(s) will fire. In the SPN we present, only immediate transitions require the specification of these probabilities, since the probability of contemporary firing for continuous (exponential) distributions is null.

5 Model of Quorum

Consider the SPN shown in figure 1 (part of a larger one considered later). At a certain moment in time, the SPN contains a token in the place labeled *START*, and a number of tokens in the places labeled *CC* (current copies), *CW* (current witnesses), *OW* (outdated witnesses), and *OC* (outdated copies), representing the number of representatives in the corresponding state at the beginning of the observation. The remaining places are empty. Assume that it is possible to reach a quorum, that is, there is a token in the *CC* place and that a minority of the hosts are down.

Transition $T16$ is enabled and can fire after an exponentially distributed amount of time signifying the time needed to send status request messages and receive responses. After $T16$ fires, the assembly of the quorum begins. It is likely that more than a majority of representatives respond; if so, a subset of them must be selected to participate in the update. As mentioned earlier, we would like to minimize the time needed to perform the update, so we prefer to include as many witnesses as possible. We must always include a current copy in the quorum, so a token is removed from the *CC* place and deposited in the *QC* (quorum copies) place when $T16$ fires. A token is also deposited in the *DRIVE* place, to start the gathering of other representatives.

Transitions $T17$, $T18$, $T19$ and $T20$ represent the inclusion of current witnesses, current copies, outdated witnesses and outdated copies in the quorum, respectively. We want these transitions to fire in the order $T17$, $T18$, $T19$ and $T20$, so they are assigned priorities of 2, 3, 4, and 5 respectively. We want these transitions to fire only until a majority is reached, so we associate an enabling function, \bar{q} with them. The logical function q evaluates to *true* if a majority of representatives has been selected to participate in the update transaction; \bar{q} is the logical complement of q . Transition

$T17$ will fire once for each current witness, then transition $T18$ will fire once for each current copy. Then outdated witnesses will be updated, and if necessary, outdated copies will be made current. These last two events requires some time, so transitions $T19$ and $T20$ are timed. As soon as q evaluates to *true*, the gathering will stop, so $T18$, $T19$, and $T20$ might not fire as many times as they are enabled.

When q evaluates to *true*, transition $T26$, with priority 6, is enabled. At this point, the tokens in places QC and QW (quorum witnesses) denote the participants in the update. The sum of the tokens in these two places represents a majority of the representatives in the file system.

We want to embed this SPN describing the gathering of a quorum within a larger SPN including failures, repairs, and the update procedure. We will consider the quorum gathering model as a subnet activated when a token is deposited in the *START* place and exited when a token is deposited in the *DONE* place. The CC , CW , OW , and OC places will be shared with the larger net. The shared places are drawn with a double-circle in all the figures. The numbers we used for the transitions correspond to one of the two instances of the subnet. The numbers for the second instance are $T21$, $T22$, $T23$, $T24$, $T25$, $T27$, respectively, instead of $T16$, $T17$, $T18$, $T19$, $T20$, $T26$.

6 The File System Model

The overall model for the distributed file system is shown in figure 2, where the contents of each of the boxes labeled “Form Quorum” is the SPN shown in figure 1, with the exception of the CC , CW , OC , and OW places, which are shared with the overall model. The left portion of figure 2 models the failure and repair of hosts where copies reside; the right portion models the failure and repair of hosts where witnesses reside; the center portion models an update transaction. We will describe each portion separately.

Places CW , DW , and OW (on the far right in the figure) represent the number of witnesses that are current, down (host has failed), and out-of-date. Transition $T2$ represents the failure of the host (we assume that all hosts are identical), transition $T4$ represents the repair of the host, and transition $T40$ represents failure of the host of an out-of-date witness. Since we assume that a witness goes out-of-date when the host fails, the output place for transition $T4$ is place OW .

A similar structure for the copies consists of transitions $T1$, $T3$ and $T39$, and places CC , DC , and OC . We assume that a copy on a host that has failed and is subsequently repaired will check for the existence of a quorum in the file system, possibly bringing itself up-to-date. This action is represented by the box labeled “Form Quorum 2”. When a host on which a copy resides is repaired, its corresponding token is deposited in the OC place, but a token is also deposited in the TRY place to represent the attempt to form a quorum. It is then possible that the out-of-date copy needs to be brought up-to-date before reaching a quorum (transition $T41$ insures that no more than one token exists in place TRY). When a token is deposited in place TRY , if it is possible to reach a quorum, then transition $T15$ fires and a token is deposited in the $START.2$ place in the SPN of figure 1. Also, a token is deposited in place $INQ.2$, which is simply used to signify that a quorum is being formed. The QC , $DONE$, and QW places in the quorum gathering SPN are the same as those shown in the overall SPN. The CC , CW , OC and OW places in the quorum gathering SPN are the same as the respective places in the overall SPN. Thus, there are arcs from these places into the boxes that are not shown explicitly in figure 2; these arcs are shown explicitly in figure 1. After a quorum is gathered and the out-of-date copy is updated (transition $T13$), the representatives are returned to their respective places (transitions $T12$ and $T14$). Transition $T13$ has higher priority than transitions $T12$ or $T14$, so that it fires before them.

The center section of the SPN models the normal update process. Transition $T0$ models the time lapse between update requests. When a request is received, a token is deposited in place REQ . If a quorum can be formed (function f evaluates to *true*), transition $T5$ will fire and deposit a token in the $START.1$ place. The box labeled “Form Quorum 1” is another instance of the quorum gathering SPN, with the $QC.1$, $QW.1$, and $DONE.1$ places in common with the overall SPN. Once the quorum gathering is completed, the remaining current representatives become outdated (transitions $T8$ and $T10$). When these two transitions have moved all the tokens from CC and CW to OC and OW respectively, $T9$ fires, allowing the update to occur. At this point, the representatives may be updated (transitions $T6$ and $T7$), or the hosts on which they reside may fail (transitions $T28$ and $T29$), after which we await the next request.

7 Manual Reconstruction of File Configuration

The SPN just defined produces a Markov chain with absorbing states, which correspond to situations where a quorum cannot be formed because more than a majority of hosts are down. Since we wanted to analyze the steady-state availability of the system, we introduce a manual intervention to reconstruct the system when a quorum is not possible. Figure 3 shows the SPN used to model the manual reconstruction of the file structure. When a request is received and it is not possible to gather a quorum (function f evaluates to *false*), transition $T30$ (in figure 2) fires and deposits a token in the $START$ place of figure 3, which we now describe. The firings of transitions $T31$ and $T32$ gather together the outdated representatives and deposit the corresponding tokens in places C and W . The failed hosts must be repaired before their representatives can be gathered; the repair is represented by timed transitions $T36$ and $T37$. When all the representatives have been gathered, transitions $T33$ and $T34$ may then begin firing, to bring representatives up-to-date. After these two transitions have emptied places C and W , transitions $T38$ can fire, putting a token in $DONE$. Transition $T35$ (in figure 2) can then remove this token and the token residing in $INREC$, restoring the initial configuration.

8 Model Specification

The model is specified to the solution package in CSPL (C-based Stochastic Petri net Language) based on the C programming language. A set of predefined functions available for the definitions of SPN entities distinguishes CSPL from C. Some of these predefined functions are *place*, *trans*, *iarc*, *oarc*, and *harc*, used for defining places, transitions, input arcs, output arcs and inhibitor arcs, respectively. There are also built-in functions for debugging the SPN and for specifying the solution method and desired output measures. The enabling, distribution, and probability functions for each transition are defined by the user as C functions; marking-dependency can be obtained using other built-in functions such as *mark*(“*place*”), returning the number of tokens in the specified place.

The enabling functions q and f , used respectively in the quorum gathering subnet (figure 1) and to trigger the reconstruction subnet (in figure 2) are logical functions defined on the current marking of the SPN. To keep these functions simple, we defined three places in the overall SPN, $INQ.1$, $INQ.2$, and $INREC$. There is a token in place $INQ.1$ if a quorum is being formed for a normal update; a token is in place $INQ.2$ if an out-of-date copy is checking to see if a valid quorum is possible; place $INREC$ holds a token during manual reconstruction of the file system.

The CSPL code describing q and f respectively is the following (in the C programming language

“&&” means “logical and” and ‘!’ means “logical not”):

```

int q() {
    if (mark(QW)+mark(QC) > mark(CC)+mark(CW)
        +mark(OC)+mark(OW)+mark(DC)+mark(DW))
        return(1); /* true */
    else
        return(0); /* false */
}

```

Thus, q evaluates to *true* iff more than half of the total number of representatives is in the places QW and QC (at least one of them is a copy, given the structure of our SPN).

```

int f() {
    if (mark(CC) && !mark(INQ.1) && !mark(INQ.2) &&
        !mark(INREC) && mark(CC)+mark(CW)+mark(OC)
        + mark(OW) > mark(DC)+mark(DW))
        return(1); /* true */
    else
        return(0); /* false */
}

```

Thus, if there is a current copy (a token in place CC), and if no other quorum or reconstruction is in progress (no token in any of places $INQ.1$, $INQ.2$ or $INREC$), and if more representatives are up than are down, the transition is enabled.

The firing rates for some timed transitions depend on the number of tokens in the input place. As an example, transition $T1$ fires at a rate equal to the failure rate of an individual host multiplied by the number of operational hosts (tokens in place CC).

```

float T1rate() {
    return(mark(CC)*phi);
}

```

where “phi” is a floating point constant representing the failure rate of a single host. The marking-dependent firing rates for the other transitions representing host failures are defined similarly.

9 Model Experimentation

For two different quorum definitions, we varied the mean time to repair failed hosts, the number of copies, and the number of witnesses. For both sets of experiments, we assumed the parameters listed in table 1 and considered the mean time to repair to be 1 hour, 2 hours, or 8 hours.

In the first set of experiments, we defined a quorum to be the majority of all representatives (a quorum must include a current copy) as previously discussed. The second set of experiments relaxed this definition by requiring a quorum to consist of a majority of representatives on *operational hosts*. For example, suppose that a total of 3 copies and 2 witnesses were created, but only one of each

is now available and current. Given the requirement that a majority of all representatives must participate in a quorum, an update cannot be serviced now. However, if we require that a quorum consist of a majority of representatives on *operational hosts*, a quorum can be formed using the available copy and witness. Under the second assumption, updates can continue until the last copy fails, then a manual reconstruction must be performed. This idea was concurrently investigated by Jajodia and Mutchler in [Jajodia87], where it was termed *dynamic voting*. A similar approach is the *adaptive voting* used in the design of fault-tolerant hardware systems [Siewiorek82]. To model adaptive voting instead of static voting, only the definitions of q and f need to be changed, to eliminate the dependency on places DW and DC .

We analyzed the availability of the system by observing the probability that a token is in place $INREC$. This measure represents the steady-state probability that the system is undergoing manual reconstruction of the file system. The availability of the file system, the complement of this probability, is shown under the two quorum definitions in table 2 for MTTR = 1, 2, and 8 hours. The model of the system with one copy had 11 states, while the model of the system with 4 copies and 3 witnesses had 1746 states.

When MTTR = 1 or 2 hours, we can see how the adaptive voting technique increases the availability if the total number of representatives is small. This technique may decrease availability if there are many representatives (see for example the configuration with 5 copies and 2 witnesses). The availability of the system decreases as copies are replaced by witnesses for both voting techniques up to a certain point, then it increases again. This behavior can be explained considering the factors affecting the overall availability.

First of all, replacing copies with witnesses decreases the total number of copies, so the probability of having all the copies down (regardless of the state of the witnesses) is higher, with a negative effect on the availability. An especially sharp decrease in the availability is experienced going from $\lceil n/2 \rceil + 1$ to $\lceil n/2 \rceil$, when n representatives are available (we present only the case n odd, see [Jajodia87] for a discussion of the case n even), the reason is the quorum policy that we are assuming. Most of the time no host is down, so, assuming $n = 7$, the quorum will contain two copies and two witnesses in the (5,2) case, and one copy and three witnesses in the (4,3) case. Having only one copy in the quorum implies that whenever that copy becomes unavailable, the other copies will be useless, because they will be out-of-date. The increase of availability after the point $\lceil n/2 \rceil$ is less intuitive, the main cause is probably the quorum policy itself (this suggests that further investigations on the quorum policies and their effect on the availability are needed).

When MTTR = 8 hours, the adaptive voting technique always increases availability although the improvement decreases as the number of witnesses increases. With 5 representatives, allowing one of the representatives to be a witness increases the availability under static voting, but decreases it under adaptive voting. With 7 representatives, the assignment of two of them to witness status maximizes availability under static voting. However, the adaptive algorithm still performs better.

Under the assumptions stated, the addition of witnesses to a fixed number of copies does not increase the availability. This can be seen by comparing the results for 3 copies and 0 witnesses (0.99908), 3 copies and 2 witnesses (0.95992), and 3 copies and 4 witnesses (0.95663) (see table 2, MTTR = 2, adaptive voting). The reason for the decrease in availability when adding extra witnesses can be explained by examining the procedure used for the selection of the participants in a quorum. The desire to speed the quorum gathering process gives preference to witnesses in the quorum, since the update procedure for a witness is fast. However, this preference increases the probability of copies being out-of-date. Since a current copy is necessary for a quorum and redundant copies are often out-of-date, a decrease in availability results.

In trying to improve the availability without considering performance, the quorum gathering procedure would give preference to copies as participants. This can be easily reflected in the SPN model under consideration. Only the priorities associated with the transitions in figure 1 need to be changed, to include current copies, outdated copies, current witnesses and then outdated witnesses, in that order. Table 3 compares the availability with 3 copies and 0, 2, or 4 witnesses with the two different preferences in the formation of the quorum using the adaptive voting algorithm. In this case, the addition of witnesses favorably affects the availability. The preference for copies in the quorum increases the time needed to perform an update, since it takes longer to write a copy than a witness. In table 4, we compare the steady-state probability of the system being in the “update” state. We can estimate this probability from the SPN by looking at the probability that a token is in place *INQ.1* (figure 2): a token is more likely to be in place *INQ.1* when preference in forming a quorum is given to copies over witnesses.

10 Conclusions

We have presented a detailed stochastic Petri net model of a replicated file system for availability analysis. The model included failure and repair of hosts, file updates, quorum formation, and manual reconstruction of the file system. The total number of representatives of a file was varied between one and seven; the composition of the representatives (number of copies and witnesses) was varied as well.

Using this model, we investigated two different voting algorithms used to maintain file consistency. The first algorithm, static voting, required a majority of all representatives to participate in an update. The second algorithm, adaptive voting, required a majority of representatives on operational units to participate in an update. In most cases, the adaptive voting algorithm resulted in a higher availability of the file system. The effect was more pronounced when the mean time to repair a host was higher, and thus a smaller number of hosts was available at a given time.

We then examined the process by which actual participants in an update are selected, given that more than a majority are available. A desire for a fast update process leads to a preference for witnesses in a quorum, with the effect of decreasing the availability of the file system. This decrease in availability can be attributed to the fact that representative not participants in the quorum become outdated, and so copies are frequently out-of-date.

To increase the availability of the file system, preference is given to copies as participants in the update, with an unfavorable effect on performance. Since it takes longer to perform an update on a copy than on a witness, it takes longer to service each request if more copies are quorum participants.

Future modeling efforts will look at the effect of converting copies to witnesses and vice versa and at the effect of using different quorum policies on the availability of the file system. We will also investigate heuristics for determining when to perform the conversion. We expect to be able to explore a wide number of policies by making minor changes or additions to the SPN we have presented.

References

- [Ajmone84] Marco Ajmone Marsan, Gianfranco Balbo, and Giovanni Conte. A class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor

- Systems. *ACM Transactions on Computer Systems*, 2(2):93–122, May 1984.
- [Barbara87] Daniel Barbara and Hector Garcia-Molina. The Reliability of Voting Mechanisms. *IEEE Transactions on Computers*, 1197–1208, October 1987.
- [Bechta84a] Joanne Bechta Dugan, Kishor S. Trivedi, Robert M. Geist, and Victor F. Nicola. Extended Stochastic Petri Nets: Applications and Analysis. In E. Gelenbe, editor, *Performance '84*, North-Holland, Paris, France, December 1984.
- [Bechta85] Joanne Bechta Dugan, Andrea Bobbio, Gianfranco Ciardo, and Kishor Trivedi. The Design of a Unified Package for the Solution of Stochastic Petri Net Models. In *Proceedings of the International Workshop on Timed Petri Nets*, Torino, Italy, July 1985.
- [Davcev85] Danco Davcev and Walter A. Burkhard. Consistency and Recovery Control for Replicated Files. In *Proceedings of the 10th Symposium on Operating Systems Principles*, pages 87–96, 1985.
- [Ellis83] Carla Schlatter Ellis and Richard A. Floyd. The Roe File System. In *Proceedings of the 3rd Symposium on Reliability in Distributed Software for Database Systems*, pages 175–181, 1983.
- [Gifford79] David K. Gifford. Weighted Voting for Replicated Data. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles*, pages 150–159, 1979.
- [Jajodia87] Sushil Jajodia and David Mutchler. Dynamic voting. *ACM SIGMOD*, 227–238, May 1987.
- [Molloy81] Michael K. Molloy. *On the Integration of Delay and Throughput Measures in Distributed Processing Models*. PhD thesis, UCLA, Los Angeles, CA, USA, 1981.
- [Natkin80] Stephan Natkin. *Reseaux de Petri Stochastiques*. These de Docteur Ingenieur, CNAM-Paris, Paris, France, June 1980.
- [Paris86] Jehan-Francois Pâris. Voting with Witnesses: A Consistency Scheme for Replicated Files. In *Proceedings of the 6th International Conference on Distributed Computing Systems*, pages 606–612, May 1986.
- [Paris86a] Jehan-Francois Pâris. Voting with a Variable Number of Copies. In *Proceedings of the Sixteenth International Symposium on Fault-Tolerant Computing*, pages 50–55, July 1986.
- [Parker83] D. Stott Parker et al. Detection of Mutual Inconsistency in Distributed Systems. *IEEE Transactions on Software Engineering*, 240–247, May 1983.
- [Peterson81] James L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.
- [Popek81] Gerald J. POpek et al. LOCUS: A Network Transparent, High Reliability Distributed System. In *Proceedings of the 8th Symposium on Operating Systems Principles*, pages 169–177, 1981.

- [Siewiorek82] D. P. Siewiorek and R. S. Swarz. *The Theory and Practice of Reliable System Design*. Digital Press, Bedford, MA, 1982.
- [Stonebraker79] M. Stonebraker. Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES. *IEEE Transactions on Software Engineering*, 188–194, May 1979.
- [Thomas79] Robert H. Thomas. A Majority Consensus Approach for Concurrency Control for Multiple Copy databases. *ACM Transactions on Database Systems*, 180–209, June 1979.

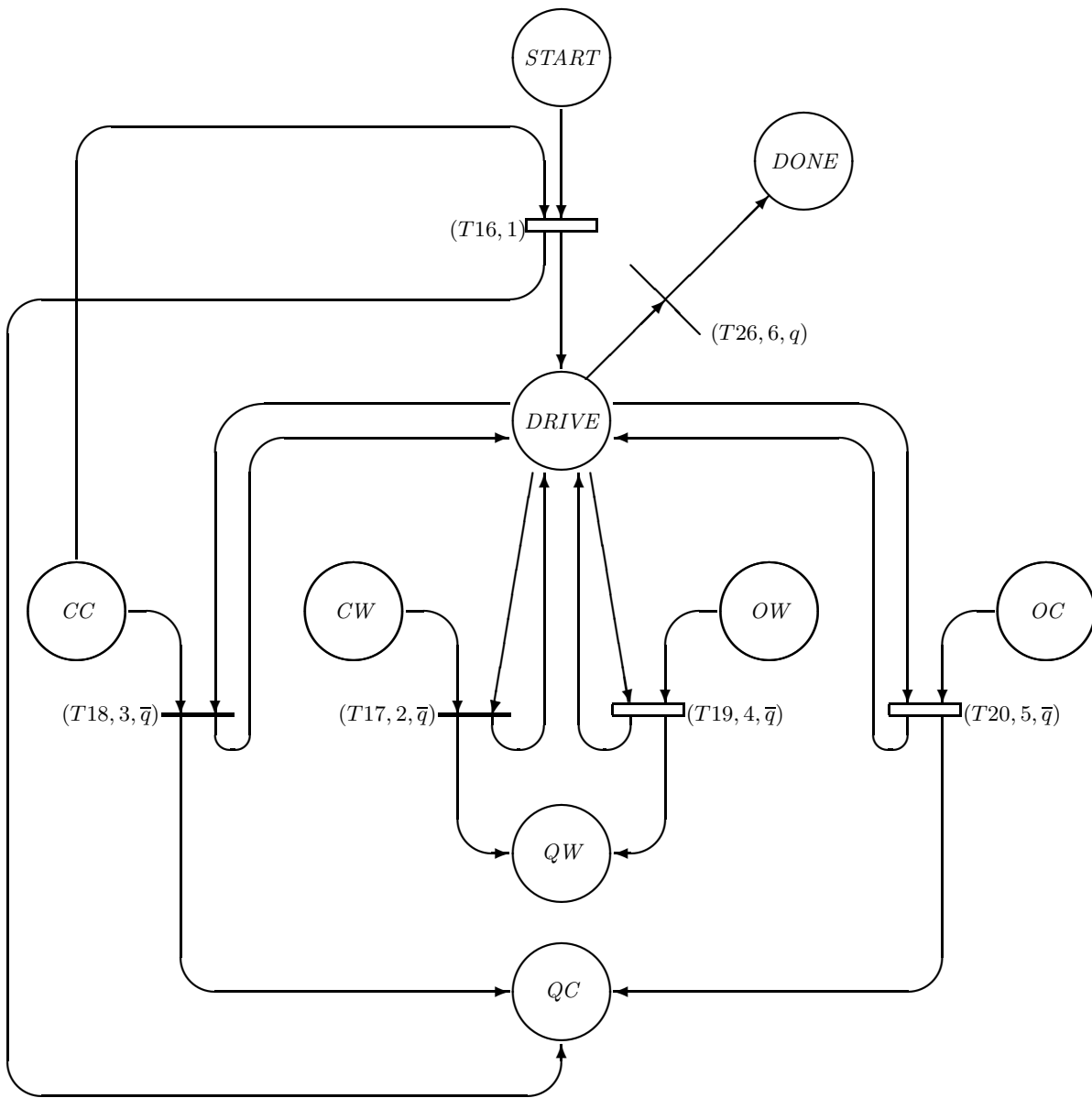


Figure 1: SPN Submodel of Quorum Gathering.

Parameter	Value (mean time)	SPN Transition(s)
time to failure for one host	100 hours	$T1, T2, T28, T29, T39, T40$
time to repair one host	1, 2, or 8 hours	$T3, T4, 38, 39$
time to get response messages	1 second	$T16, T21$
time to update witness	50 msec.	$T7, T19, T24$
time to update copy	500 msec.	$T6, T20, T25$
time to reconstruct file system	1 hour	$T33, T34$
time between requests	1 hour	$T0$

Table 1: Parameters used in Model Experimentation.

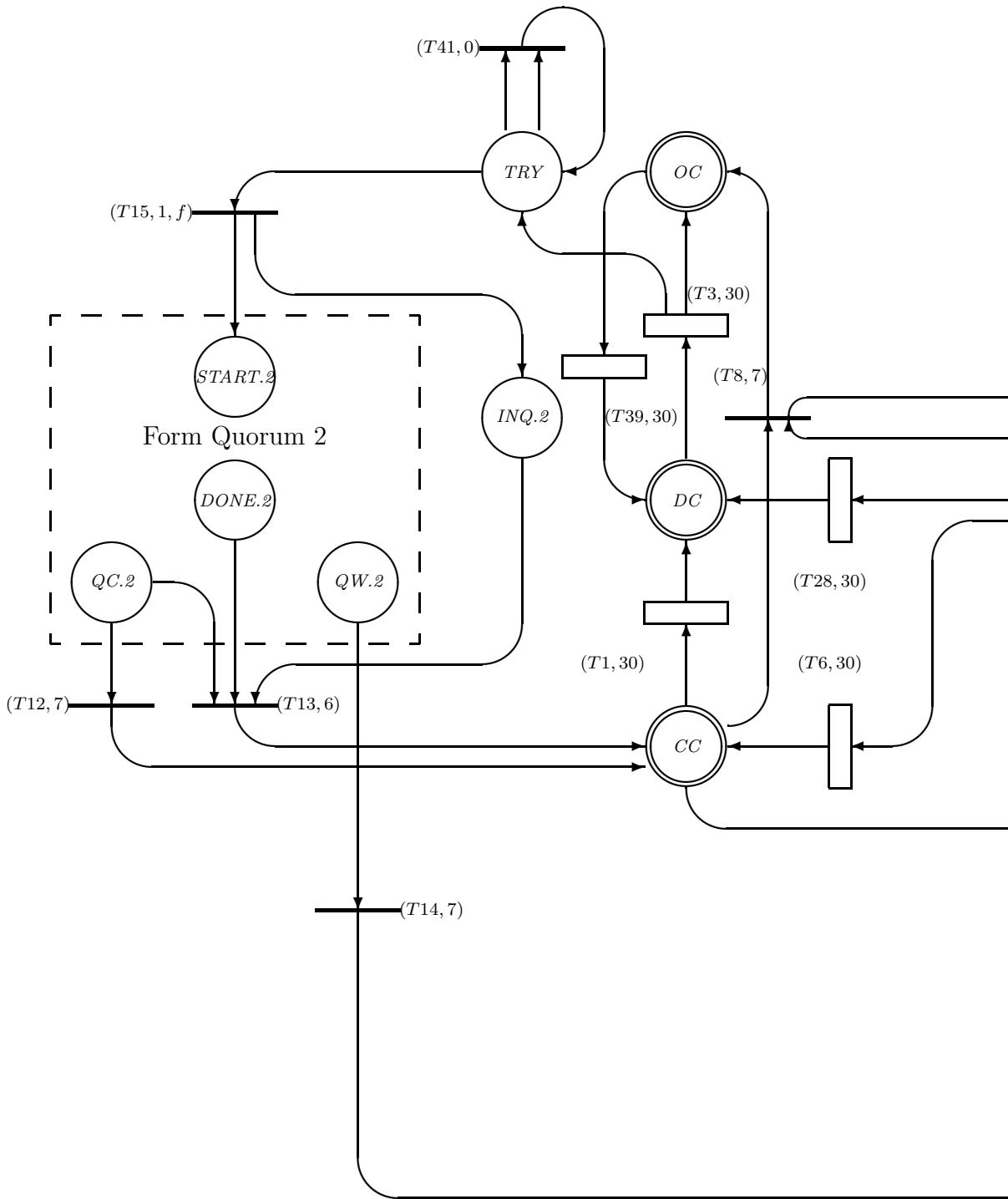


Figure 2: Overall model of replicated file system.

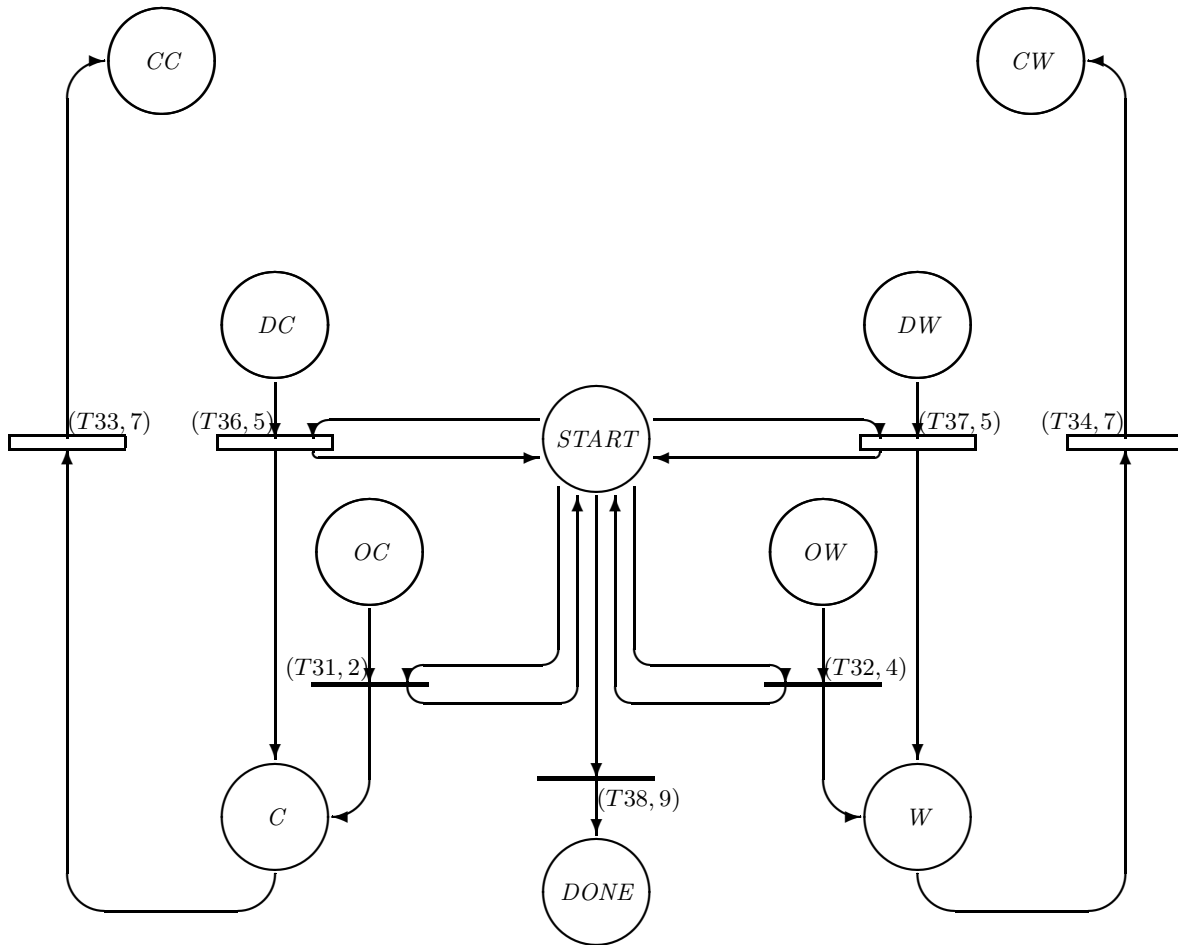


Figure 3: SPN Submodel of Manual Reconstruction of File System.

Configuration		File system availability	
Number of copies	Number of witnesses	Majority of ALL (static voting)	Majority of UP (adaptive voting)
MTTR = 1 hour. Single file availability = 0.99.			
1	0	0.985358	0.985358
3	0	0.998627	0.999567
2	1	0.975953	0.976184
1	2	0.982280	0.982643
5	0	0.999817	0.999989
4	1	0.999401	0.999453
3	2	0.967340	0.967355
2	3	0.974384	0.974393
1	4	0.979079	0.979138
7	0	0.999967	1.000000
6	1	0.999970	0.999987
5	2	0.999386	0.999331
4	3	0.958971	0.958938
3	4	0.965660	0.965599
2	5	0.972123	0.972120
1	6	0.975113	0.975106
MTTR = 2 hours. Single file availability = 0.98.			
1	0	0.977397	0.977397
3	0	0.995328	0.999084
2	1	0.967400	0.968444
1	2	0.972957	0.974324
5	0	0.998711	0.999964
4	1	0.998374	0.998823
3	2	0.959764	0.959924
2	3	0.965618	0.965774
1	4	0.969970	0.970407
7	0	0.999548	0.999998
6	1	0.999704	0.999954
5	2	0.998814	0.998495
4	3	0.952103	0.951944
3	4	0.956874	0.956631
2	5	0.963034	0.963075
1	6	0.965789	0.965896
MTTR = 8 hours. Single file availability = 0.926.			
1	0	0.925640	0.925640
3	0	0.947057	0.990715
2	1	0.901961	0.917182
1	2	0.902532	0.917441
5	0	0.953294	0.998217
4	1	0.965207	0.984715
3	2	0.899270	0.908415
2	3	0.894530	0.903121
1	4	0.892000	0.906290
7	0	0.950902	0.999452
6	1	0.966146	0.996002
5	2	0.974166	0.977000
4	3	0.897530	0.899770
3	4	0.886600	0.888120
2	5	0.883620	0.892050
1	6	0.877730	0.891830

Table 2: Replicated file system availability.

Configuration		File system availability (MTTR = 2 hours)	
copies	witnesses	prefer witnesses	prefer copies
3	0	0.99908	0.99908
3	2	0.95992	0.99982
3	4	0.95663	0.99982

Table 3: Comparison of file system availability when preference is given to witnesses *vs.* copies in forming quorum.

Configuration		Probability of being in the process of forming a quorum (MTTR = 2 hours)	
copies	witnesses	prefer witnesses	prefer copies
3	0	0.000565	0.000565
3	2	0.000414	0.000711
3	4	0.000416	0.000718

Table 4: Comparison of steady-state probability that a quorum is being formed where preference is given to witnesses *vs.* copies.