Adaptive Distributed Resource Allocation and Diagnostics Using Cooperative Information-Sharing Strategies

Partha S. Dutta, Nicholas R. Jennings, and Luc Moreau
School of Electronics & Computer Science
University of Southampton
Highfield, Southampton SO 17 1BJ, U.K.
{psd,nrj,l.Moreau}@ecs.soton.ac.uk

ABSTRACT

A major challenge in efficiently solving distributed resource allocation problems is to cope with the dynamic state changes that characterise such systems. An effective solution to this problem should be able to detect state changes and determine why they occur (diagnosing the cause) in order to adapt to the prevailing situation. Now, since agents typically have localised views and communication constraints that prohibit global instantaneous synchronisation, we argue that cooperative information-sharing can provide them with the necessary adaptiveness and diagnostics ability. To this end, we develop a novel information-sharing algorithm for resource allocation tasks by building upon the most effective algorithm currently available in this domain. Then, using empirical analyses on a resource allocation application with dynamic state changes, network call routing with network failures, we show that, compared to the benchmark, our new algorithm achieves up to a 20% increase in call throughput, up to 3.5 times faster throughput recovery after failures, and provides a novel mechanism for distributed failure diagnosis without false positives and false negatives.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Experimentation, Theory

Keywords

Resource allocation, state diagnosis, information-sharing

1. INTRODUCTION

Resource allocation is a core functionality of many distributed systems [5, 8]. In such systems, multiple agents typically allocate resources under a decentralised regime using their individual localised views of the system, in order to complete their tasks. To do so, they use estimates of system states to ensure their individual actions are coordinated [2]. Now, to do this effectively in complex environments (where states change dynamically), such estimation necessarily involves detecting where, when, and why changes occur. Thus, to maintain these estimates and to perform such diagnostics, the agents need to share information. In particular, they need to communicate their individual views of the system to one another. Such information can then be used to generate robust local estimates and allow the agents to adapt these when states change. To this end, we develop and empirically evaluate a novel information-sharing algorithm that allows distributed agents to adapt dynamically to state changes and to perform distributed state diagnostics.

Given its importance, several strands of work on cooperative MAS have focused on analysing how communication can generate better solutions to distributed coordination [1, 6, 7]. However, in these, several limitations exist. For example, in [7], the authors develop a theoretical model for optimising information-sharing with the assumption that the agents need to synchronise their views periodically. In a typical distributed application, however, the agents should be able to act in a decentralised fashion without requiring to synchronise. The work in [6] considers how communication can improve the expected utility of an individual agent. We argue, however, that in cooperative systems, communication should be targeted towards improving overall system performance. Finally, the partial global planning algorithm in [1] allows the agents to communicate partial plans, representing their individual views of the system, to ensure consistent global solutions. Nevertheless, this work uses a set of pre-determined coordination mechanisms which can be inadequate for the agents to adapt to all possible environment dynamics that cannot be predicted a priori.

In our previous work, we addressed these limitations by developing an information-sharing protocol (the *post task-completion* (PTC) protocol) for distributed resource allocation [3]. Specifically, we showed that such cooperative information-sharing can generate highly effective distributed resource allocation. Our algorithm outperformed a range of state-of-the-art algorithms (including Littman's Q-routing, Stone's TPOT-RL, and a broadcast protocol) that have previously been shown to perform highly effective distributed task processing in dynamic systems. In [3], we chose an exemplar resource allocation application, that of call routing in resource-constrained mesh networks, to verify the performance advantage of our solution in a real application and demonstrated that our solution is more generally applicable to all such problems using a formal analysis. However, PTC did not produce the adaptiveness required in highly dynamic resource allocation problems by ensuring effective state diagnostics. Thus, in this paper, we extend PTC (to e-PTC) to remove these limitations. This necessarily implies an improvement over the benchmarks used in [3].

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.

In more detail, e-PTC allows the agents to share information both when they successfully complete tasks and fail to do so. In contrast, PTC only allows agents communicate state information after they successfully complete tasks. In so doing, they are able to communicate their local states and build state estimates when the system is functioning under "steady state". However, unpredictable incidents such as process failures, network node saturation, and hardware crashes, can lead to subsequent failures in task processing. To adapt under these conditions, which are common in many distributed systems [10], the agents need to maintain estimates of the dynamic state changes and diagnose states for efficient recovery. Accordingly, we argue that such capabilities can be incorporated by allowing information-sharing after task failures. Indeed, by developing e-PTC, this paper extends the current state-of-the-art in the following significant ways: (1) It provides a simple but effective means of providing highly adaptive behaviour among distributed agents in dynamic environments. (2) It establishes, using empirical analyses, the advantage of e-PTC in providing effective resource allocation in a real application: call routing in mesh networks. Specifically, e-PTC generates up to 20% higher call throughput than PTC when network failures occur and allows up to 3.5 times faster recovery of throughput after failures. (3) A novel state-diagnosis algorithm is developed, based on e-PTC, that allows distributed agents with localised views to diagnose network failures correctly (no false positives) and only when failures occur (no false negatives).

In the rest of this paper, section 2 briefly describes the distributed resource allocation task in the context of our application domain and reviews how information-sharing aids state estimation. Then, section 3 describes e-PTC and how it generates effective adaptiveness in highly dynamic systems. The experimental setup is described in section 4 and the results in section 5. Section 6 develops the state-diagnosis algorithm using e-PTC and presents empirical results on its performance. Section 7 concludes.

2. INFORMATION-SHARING IN RESOURCE ALLOCATION TASKS

The task of decentralised routing in mesh networks requires individual agents to allocate node bandwidth (representing the resources in the system) and forward a call to a neighbouring node (the next hop along the call route) such that the call reaches the destination when a set of end-to-end bandwidth is allocated. To forward a call, an agent chooses that neighbour for which it estimates that the call would be placed via the most efficient overall route. Thus, each agent maintains an estimate of the bandwidth availability along paths to any other node. This reflects its partial view of the overall network. It attempts to keep these estimates up-to-date as much as possible by using the information exchanged between agents. The more up-to-date the estimates with respect to the actual network states, the higher the quality of routing and, thus, the better the network performance.

More specifically, each node i in a network of A nodes, maintains a routing table RT_i, where each element $RT_i(d,n)$ ($d \in A$ is the destination node, and $n \in A$ is a neighbour of i) represents i's estimate of the best end-to-end bandwidth availability in going from n to d. Node i chooses a neighbour j using a Boltzmann distribution over its neighbours based on the RT_i values: $e^{\frac{-RT_i(d,j)}{\tau}} / \sum_n e^{\frac{-RT_i(d,n)}{\tau}}$, where τ is the "temperature" parameter used to set the skewness of the distribution. When a call forward request reaches the destination, the latter sends "upstream" along the call route an `ack` message indicating a successful call connection. Otherwise, if a call forwarding fails before reaching the destination, the node at which it fails sends a `drop` message "upstream". As each node i transmits the `ack` or `drop`, it appends its state value s_i to the message. In this application, a node's state is the bandwidth available at that time expressed as a fraction of the maximum available bandwidth.[1] Therefore, a node (say, k) receiving an `ack` or `drop` from the immediate "downstream" node (say, l) gets the set of state values that were appended by all "downstream" nodes on this call route. These state values are then used by k to update its RT as: $RT_k(d,l) \leftarrow (1-\alpha)RT_k(d,l) + \alpha\Gamma(s_l, ..., s_d)$.[2] Here, $\Gamma()$ is an aggregation function on a set of node state values, and α is a discount factor ($0 < \alpha < 1$) that puts different weights on previous and current information. In particular, we choose Γ as the *minimum* of the set of node states since the node with the minimum available bandwidth determines the maximum number of calls that can be accommodated along a given route. Figure 1 presents a schematic of the above description.

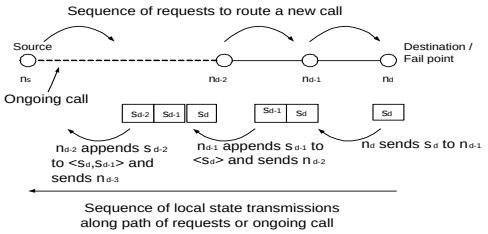


Figure 1: Call setup and information exchange

In this system, state information is exchanged between nodes *after every call connection and failure*. This is the definition of e-PTC (see [2] for more details of the design of e-PTC on this application). Note, the original PTC allowed information-sharing only after a call connection. Thus, e-PTC expands the set of events that PTC uses for sharing information.

3. ADAPTIVENESS IN HIGHLY DYNAMIC SYSTEMS

The e-PTC algorithm dictates that agents should share information after both successful task completions and task failures. Sharing information after task completion provides e-PTC with the same advantage as that of PTC: effective resource allocation when the system states are steady. However, when unforeseen events such as network failures cause task processing to fail, e-PTC allows the agents to share state information and thus allows for the estimation of the state changes caused by such events and diagnosis of the events. These capabilities cannot be achieved using PTC alone. Thus, the e-PTC algorithm, together with the estimate-generation mechanism discussed in section 2, provide effective adaptive behaviour in dynamic systems which is a significant improvement over the current state-of-the-art. In the rest of this section, we first describe the model we adopt to define network failures that cause dynamic state changes and, then, describe the design of e-PTC in the context of call routing with failures.

3.1 Model of Dynamic State Changes

Network failures cause dynamic state changes in our application. In the failure model, there are the following components:

[1] As nodes route new calls or existing calls terminate, available bandwidth (hence, node state) varies dynamically.

[2] This mechanism of computing a discounted running average is used to learn estimates in environments where the dynamics are not known a priori [11].

Failure Characterisation: Only **stop** failures are considered because these are the most commonly studied in distributed systems [9] and, hence, form a good testbed for measuring the performance of e-PTC. We assume that a failed node (1) cannot run any processes, (2) cannot communicate with other nodes, and (3) once failed, continues to remain so. Thus, after a failure, all ongoing calls on a node will be lost and it will not be able to detect or acknowledge new call forward requests. A failure would cause both bandwidth availability (by removing existing calls) and the topology to change, thereby, changing the network state.

Detection Characterisation: After a node fails, one of its neighbours actually detects that it has failed after the latter makes a forwarding request. The requesting node detects the failure status since it does not receive any response to its request. It then generates `drop` messages to inform its neighbouring nodes to deallocate the reserved bandwidth for the calls that were ongoing along this affected route.

3.2 Adaptiveness and Diagnostics with e-PTC

As identified in section 3.1, a failure causes network states to change dynamically. The e-PTC algorithm, by distributing state information after failures, allows the agents to assess these state changes. This, in turn, lets the agents adapt their task processing to best respond to state changes and to diagnose where in the system the failure has occurred.

In more detail, figure 2 shows the schematic of a node failure and the subsequent transmission of state information by its neighbour. In this figure, node f is shown to have failed. One of its neighbours, i, detects that f has failed after it attempts to forward a call request to f and does not get a response. Thus, it deallocates its bandwidth for all calls that were being routed through both i and f, and that for the new call it was trying to set up. Also, it informs each j (a neighbour of i that is also on the path of calls via i and f) by sending a `drop` message to de-allocate bandwidth. Each such j continues to propagate the `drop` message until the terminal node on that path is reached. Note that, a similar process is undertaken by each neighbour of f whenever they detect that f has failed. In the above process, after receiving a `drop` message, a node updates its estimate (see section 2 for the estimate-update mechanism) about the affected network path from the node state values appended to the message by each agent transmitting it. In this manner, the nodes learn that the bandwidth availability along the affected path has changed. Such information, therefore, allows them to adapt their routing behaviour to best respond to the state dynamics.

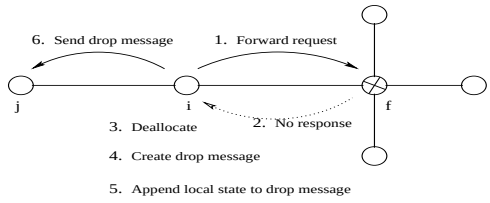


Figure 2: Network failure detection schematic

Importantly, using this information, a node can diagnose if a certain node in the network has failed. Note, when i transmits state information with a `drop` message, it assigns a value of zero for f's state. Alternatively, when a node has no available bandwidth, it rejects a forwarding request and, therefore, the call attempt fails. Similar to a node failure, the state value transmitted with a `drop` message in this case is also zero. Therefore, for correct failure diagnosis, an agent should be able to distinguish between the information (zero) received from a node that has failed and that is saturated (also zero). The key to do this is to observe a *sequence* of state values. Note, to indicate a failure, a separate message (e.g., `fail`) could be used instead of a zero state. However, the state values are used to update the routing tables. Hence, a separate message to explicitly indicate failure would not be useful for state estimation. In fact, as shown in section 6, the zero state values can be used for diagnosing failures in addition to state estimation. In section 6, we develop an algorithm that computes the *number of consecutive state values* an agent needs to monitor from another node to achieve correct diagnosis. It is shown that this algorithm is both (i) without false positives — a failure is diagnosed only if one has occurred, and (ii) without false negatives — once a failure occurs, a positive diagnosis is guaranteed.

4. EXPERIMENTAL SETUP

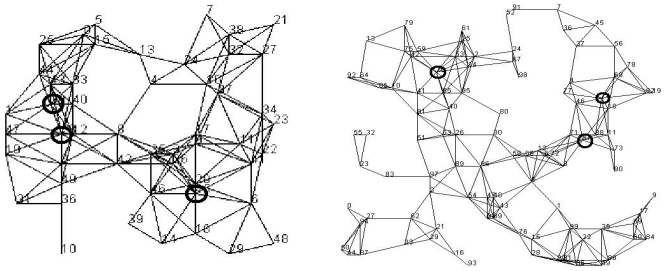
This section first outlines the simulation of a limited-bandwidth mesh network application used for call routing with failures. This system effectively emulates distributed resource allocation tasks under highly dynamic environments (see [2] for additional justification). Then, we go on to present the criteria used to evaluate the performance of e-PTC.

4.1 The Simulation Environment

Each node has a fixed maximum bandwidth (C units). This captures the limited bandwidth of mesh networks. Each call takes one bandwidth unit. The probability (p_c) of a call originating at any node is termed the "load". Call destinations are selected uniformly. A node reserves one bandwidth unit for a call after forwarding it to a neighbour. The reservation is removed if the call fails to connect or terminates after the duration t_l, for which the call lasts, measured since it is routed to its destination. Before forwarding, an agent inserts its id to the call so that a call has a sequence of ids used to transmit state information after a connection or failure. If connected, an acknowledgement (`ack`) message is transmitted from the destination up to the source. In case of a failure, the failed node's neighbour transmits a `drop` message along the call path and bandwidth reserved for this call is de-allocated. Nodes share state information by appending state values to an `ack` or `drop` message: only `ack` is used by PTC, while e-PTC uses both. Numerically, a node state is the bandwidth units available expressed as a fraction of the maximum state estimate (C). Thus, if it has x units available, its state is $\frac{x}{C}$. The routing tables are initialised to 1.0, the maximum availability. One simulation run consists of 2×10^6 simulated time steps (time is measured on a global clock) where in one step any agent can send one message to one of its neighbours. Experiments are run for a sufficient number of runs to ensure statistical significance at the 95% confidence level. The following parameter values are chosen: a network with a total of $N = 50$ nodes (figure 3(a)), and $N = 100$ nodes (figure 3(b)), $C = 10$, $\alpha = 0.03$, $\tau = 0.1$, $p_c = \{0.1, 0.2, 0.4, 0.6, 0.8\}$ (to test the effect of different loads), and $t_l = 50$ for the 50-node network and $t_l = 100$ for the 100-node network. Experiments have been conducted on several other topologies and similar trends are observed in the results [2]. Here, therefore, we report results from these sample topologies. Also, the same general observations hold for other values of α and τ.

4.2 Performance Evaluation Criteria

The following criteria are used to compare the performance of e-PTC against that of PTC to determine its adaptiveness in dynamic resource allocation tasks.



(a) 50-node topology (b) 100-node topology

Figure 3: Network topologies

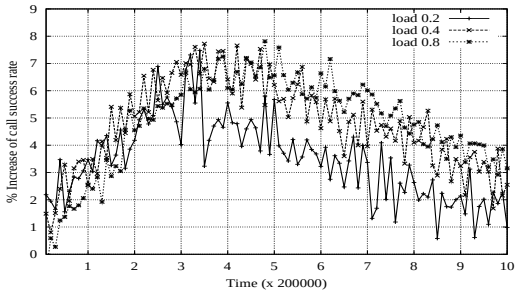
Call Success Rate (CSR): The advantage of having a highly adaptive solution in dynamic systems is to have a high rate of successful task processing. In the call routing application, this is measured in terms of the call success rate. Specifically, if, in a given time interval, K calls originate out of which k are successfully connected, the CSR within that period is: k/K. The higher the CSR in a given system, the better the performance of the information-sharing algorithm by maintaining high-quality routing tables.

Recovery of Call Success Rate: Good adaptiveness in dynamic systems should allow quick recovery of performance after state changes occur. Now, in our application, after a failure, the CSR will necessarily drop since a set of network paths become unavailable. However, as the agents adapt their estimates and find alternative routes, the CSR should recover from the initial drop. The efficiencies of e-PTC and PTC are compared by measuring how fast the CSR recovers following failures. Thus, we compare the CSR (denoted as r_{fail}) when a node fails in the simulation (termed the "test" case) to the CSR (r_{miss}) when the corresponding node is removed from the topology from the start of the simulation (termed the "baseline" case) keeping all other parameters identical. In the baseline, the agents learn the routing tables based on the topology that results after failure in the test simulation. Now, if all agents had global information, then, after failure, r_{fail} could recover and become equal to r_{miss}. But, since the agents do not have this, r_{fail} is unlikely to be able to recover to exactly r_{miss}. Nevertheless, the closer and faster it can get to r_{miss}, the more efficient the system.

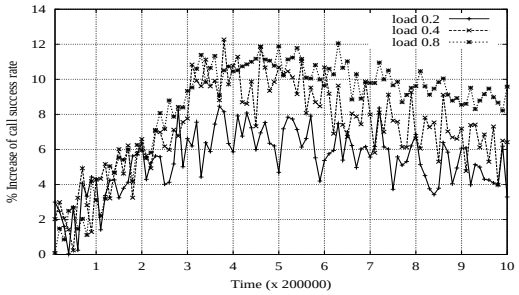
Message Size (MS): Any advantage of e-PTC over PTC in CSR or CSR recovery is due to sharing more information. Although one message is transmitted for a call success (`ack`) and one for a call failure (`drop`) in both e-PTC and PTC, these may not be of the same size. In both cases, the message size is determined by the number of state values appended. Thus, the average size of k messages is $\sum_{i=1}^{k}(l_i)/k$, where l_i is the size of the i^{th} message. To assess if e-PTC has an overhead over PTC, we compare the average size of messages generated in both.

5. RESULTS AND ANALYSIS

In this section, first, the CSR and MS of e-PTC and PTC are compared in the absence of failures to test if e-PTC has any inherent advantage (a higher CSR) or overhead (higher MS). The second set of experiments then measure their performances when dynamic state changes occur due to failures. Since it has already been shown that PTC performs better than a range of benchmark algorithms widely used for adaptive distributed resource allocation tasks [3], any improvement in e-PTC over PTC necessarily indicates an improvement over the current state-of-the-art.



(a) % CSR increase in 50-node topology

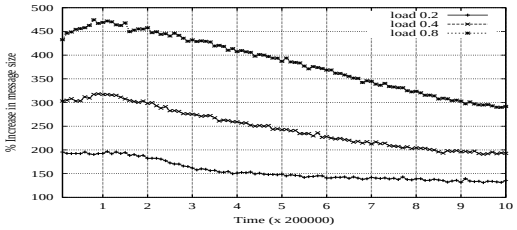


(b) % CSR increase in 100-node topology

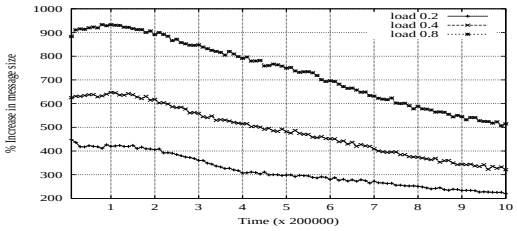
Figure 4: CSR increase in e-PTC over PTC without failures

5.1 No State Change Dynamics (No Failures)

The time-variation of the network call success rates for e-PTC and PTC are compared by calculating the percentage deviation of the e-PTC CSR from that of PTC: $(\texttt{CSR(ePTC)} - \texttt{CSR(PTC)})/\texttt{CSR(PTC)}$. These values are plotted for different network load values in figure 4. Now, the motivation for designing e-PTC is to generate effective adaptation of the system in response to dynamic state changes. However, the results in figure 4 show that *even without such dynamic state changes, e-PTC has an advantage over PTC in terms of generating higher CSR*: in figure 4(b), at load 0.8, e-PTC achieves a peak 11% increase in CSR compared to PTC. When no failures occur in the network, the source of state-change dynamics is due to network saturation when some nodes are unable to transmit further calls. Now, this obviously happens more at higher loads. Thus, the call success rates of both PTC and e-PTC decrease with increasing load. Note, in e-PTC, nodes share information after call failures. Thus, with increasing load and, hence, more call failures due to network saturation, e-PTC allows more information-sharing and ensures that the routing table estimates remain more up-to-date with the actual bandwidth availability. This allows e-PTC to adapt to the dynamics caused due to saturation better than PTC. The CSR of e-PTC does not decrease as much as that of PTC with increasing load; in figure 4, we observe the relative improvement of e-PTC CSR over PTC CSR to increase with increasing load.



(a) % MS increase in 50-node topology



(b) % MS increase in 100-node topology

Figure 5: MS increase in e-PTC over PTC without failures

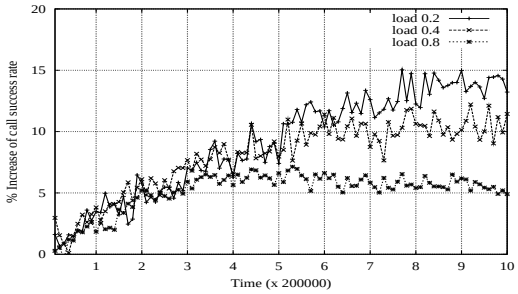
The average message size of e-PTC and PTC are compared in an analogous manner in figure 5. These results show that e-PTC incurs more message overhead than PTC and it increases with load. So, in figure 5(a), the e-PTC MS is about 2.3 times of PTC MS at load 0.2, and it is about 3.9 times at load 0.8. Now, e-PTC communicates state information when calls fail to keep the routing tables updated. However, this causes the MS (of the `drop` messages) to increase. This effect becomes more pronounced at higher loads when more calls fail (as explained before). Therefore, e-PTC attains a higher CSR at a cost of larger MS compared to PTC.

5.2 With Dynamic State Changes (Failures)

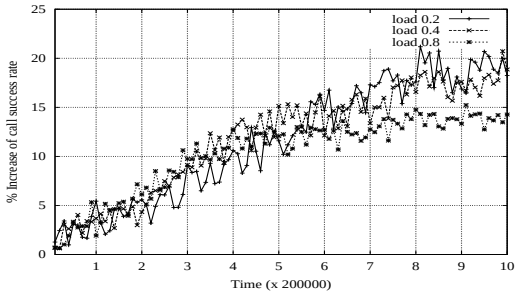
State-change dynamics are incorporated in our simulator by way of network failures. We allow one node to fail at a given time and different failures to occur at equidistant points in an experiment. More arbitrary failure patters (e.g., several failures occurring at close intervals) have been used in our ongoing work that show similar broad trends. The nodes that fail are pre-selected. However, different combinations of nodes are selected to fail in different experiments. Note, the failures are initiated by the simulator and no agent knows when and where one would occur. The results reported here use failures of (i) 3 nodes with the highest edge connectivity (encircled in figure 3), and (ii) 3 randomly chosen nodes with average edge connectivity. The motivation for choosing (i) is that the nodes with the most connectivity are critical points and heavily used for routing. Thus, their failures cause a major disruption in the network state which provides a "hard" test for the advantage, if any, that e-PTC has over PTC. However, (ii) provides a more typical scenario. We report results using (i) since the results from (ii) show identical trends.

5.2.1 Call Success Rate.

The percentage increase of call success rate of e-PTC over PTC with the three most-connected nodes failing are shown in figure 6. As explained in section 5.1, the CSRs of both e-PTC and PTC decrease with increasing load. There is an additional drop due to node



(a) % CSR increase in 50-node topology



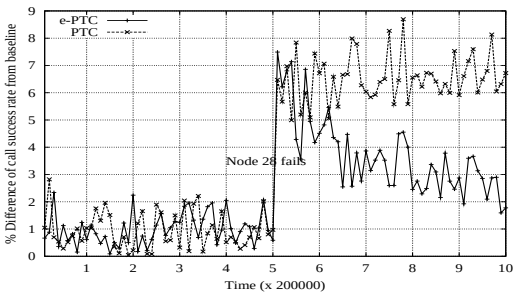
(b) % CSR increase in 100-node topology

Figure 6: CSR increase in e-PTC over PTC with 3 most-connected nodes failing

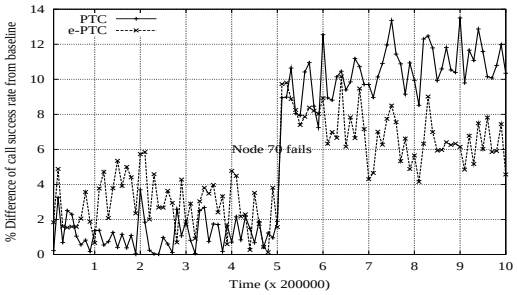
failure since it results in less bandwidth being available. However, *the CSR for e-PTC is always higher than that for PTC, confirming the effectiveness of e-PTC over PTC*. The difference in their performances is, however, more pronounced at lighter loads because at high loads the effects of both saturation and failures offset the advantage achieved by e-PTC over PTC (unlike section 5.1 where a higher relative improvement was observed at higher loads). Thus, in figure 6(b), after all 3 nodes fail, the CSR of e-PTC is about 20% higher than PTC at load 0.2 and about 14% at load 0.8. Moreover, *at a given load, the relative improvement in CSR of e-PTC over PTC increases with the number of failures*. Hence, although the CSRs decrease with failures, *the rate of deterioration in e-PTC is lower than in PTC*. These results indicate that our e-PTC algorithm generates better adaptiveness in response to state changes caused by failures. This is the reason for the networks to achieve higher call success rates with e-PTC compared to PTC in such dynamic conditions.

5.2.2 Recovery Rate.

To evaluate the efficiency with which e-PTC and PTC allow the network call success rate to recover after failures occur, we compare the CSRs of the e-PTC(test) and PTC(test) strategies against that of the baseline strategy (see section 4.2 for the definitions of these). Specifically, the percentage difference between the CSRs of e-PTC(test) and the baseline, and that between the CSRs of PTC(test) and the baseline are computed as: $| \mathtt{CSR}_{\mathtt{base}} - \mathtt{CSR}_{\mathtt{test}} | / \mathtt{CSR}_{\mathtt{base}}$. Figure 7 shows these results, generated when the highest-connected node in the corresponding networks fail halfway through the simulation and a load of 0.1 is used. Similar trends have been observed with all other load values. Now, before the failure, e-



(a) CSR recovery in 50-node topology



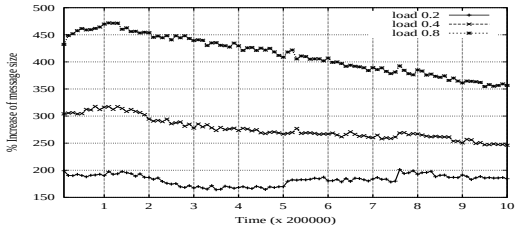
(b) CSR recovery in 100-node topology

Figure 7: Recovery of CSR after failure

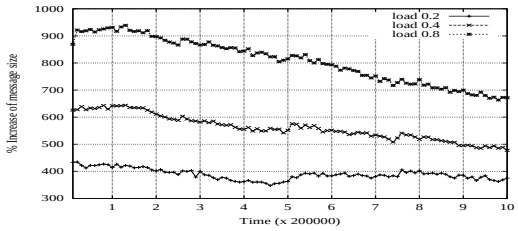
PTC(test) has the highest CSR (similar to section 5.1). However, after the failure in the test cases, the CSRs of both e-PTC(test) and PTC(test) drop. However, in figure 7, the CSR of e-PTC(test) recovers at a faster rate towards the baseline than PTC(test). The percentage deviation of the CSRs in both sets of graphs of figure 7 are observed to decrease faster for e-PTC than PTC. For example, in figure 7(a), at the end of a simulation run, e-PTC(test) CSR reaches within about 2.0% of the baseline CSR which is 3.5 times (350%) better than that of PTC(test) which reaches within 7% of the baseline. These results, therefore, confirm that e-PTC ensures a *superior efficiency in the recovery of system performance after state-changes occur.* Thus, better adaptiveness to dynamic environments is indeed achieved by e-PTC than PTC.

5.2.3 Message Size.

Sections 5.2.1 and 5.2.2 have confirmed that using e-PTC, distributed agents with localised views can achieve significantly better adaptiveness when states change dynamically. This advantage of e-PTC is generated by sharing information about the state changes, a feature lacking in PTC. Thus, the message overhead of e-PTC exceeds that of PTC. Specifically, figure 8 shows the percentage increase in message size of e-PTC compared with that of PTC, computed in a way analogous to the results in figure 5. However, the results in figure 8 are generated when the three highest-connected nodes in the corresponding topologies fail. These results show that MS *of e-PTC is significantly higher than PTC*. Also, they show that the higher the load, the greater is the increase in the MS of e-PTC relative to PTC: in figure 8(a), at the end of a simulation run, e-PTC MS is about 2.8 times that of PTC at load 0.2, whereas it is about 4.6 times at load 0.8. As mentioned earlier, e-PTC incurs the extra overhead in message size since it uses `drop` messages



(a) % MS increase in 50-node topology



(b) % MS increase in 100-node topology

Figure 8: MS increase with 3 most-connected nodes failing

to communicate the state-change information. Such information-sharing, while allowing the routing tables to be more up-to-date compared to PTC, leads to larger messages in e-PTC. As the number of call failures increase with load and network failures, so does the number of state-value transmissions in `drop` messages. One way of reducing the message overhead of e-PTC is to selectively communicate information. We are developing such a mechanism in our ongoing work, where agents use their communication histories to judge how useful a certain information would be to a recipient. Significant savings in message overhead are observed while maintaining the advantages of e-PTC.

6. STATE DIAGNOSIS USING E-PTC

The results of section 5 indicate how e-PTC ensures that agents *adapt* effectively to dynamic state changes caused by failures. In this section, we elaborate on the additional benefit of e-PTC in *diagnosing* such state changes. In particular, we develop an algorithm, based on the shared information generated by e-PTC, that distributed agents can use to detect if a certain node has failed. Such a capability would then be useful for automating distributed diagnosis and recovery.

Here, failure diagnosis means that a node raises an alert when it interprets, from the state values it receives along `drop` messages, a fail condition of another node. The detecting node is termed the "monitoring" node and the one interpreted is the "monitored" node. Note, any node can take either of these roles because information is distributed between all nodes in the network. This is because calls can originate from and terminate at all nodes and, thus, e-PTC would distribute information along call paths between all node pairs. Our aim in this context is to ensure that failure diagnosis using e-PTC is both truly indicative of a failure (no false positives), and that all failures will be detected (no false negatives). To ensure the first property, the monitoring node needs to observe a series of state values of the monitored node (explained shortly). The second property guarantees a detection.

To explain the diagnosis process, a node n_a is considered to do the monitoring of another node n_b (the monitored node). Let T be the average interval at which n_a receives the state values of n_b, distributed by e-PTC. The state values of n_b can be transmitted along `ack` and/or `drop` messages either when n_b is saturated or unsaturated, or when the latter has failed. Now, the state values of a node that is saturated and that is failed are both zeros (section 3.2). However, the difference is that a saturated node becomes unsaturated when bandwidth becomes available, whereas a failed node continues to remain so. Therefore, after receiving the first zero state value of a saturated (not failed) n_b, if n_a continues to observe the state values, then it will receive a non-zero state value after a *certain number of observations*. This will happen when n_b becomes unsaturated. Against this background, note that if n_a has the information about the average durations for which n_b remains alternately saturated and unsaturated, then it can compute the number of such observations using its knowledge of T. We assume that n_b remains alternately saturated for an average period of $\hat{t}_s$ and unsaturated for $\hat{t}_{us}$. To summarise, when n_a receives a zero state value of n_b, it assumes n_b is saturated. Then, using the knowledge of T, $\hat{t}_s$, and $\hat{t}_{us}$, n_a computes *the minimum number of observations* (termed as m) *after which it would receive a non-zero state value* of n_b. If the actual number of zero-state values of n_b received exceeds the computed value, then n_a interprets a failure by rejecting its previous assumption of a saturated n_b. The algorithm to compute m is detailed in the following section. Now, out of T, $\hat{t}_s$, and $\hat{t}_{us}$, n_a can estimate online the average interval (T) at which it receives information from n_b. However, it cannot estimate online $\hat{t}_s$ and $\hat{t}_{us}$ of n_b since it does not have information about the overall network load. But, n_a can acquire these estimates from node traffic statistics that are usually available for most networks [4].

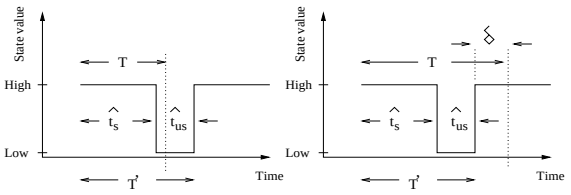
6.1 The Failure Diagnosis Algorithm

After receiving the first zero state of n_b, n_a invokes the algorithm assuming the state value to be from a $\hat{t}_s$ interval of n_b (i.e., from a saturated n_b; it may of course be from a failed n_b). Now, knowing that it would receive state values from n_b after every T units and that n_b would remain alternately saturated for $\hat{t}_s$ and unsaturated for $\hat{t}_{us}$, n_a determines whether the next reception should be from a $\hat{t}_s$ or a $\hat{t}_{us}$ interval of n_b. This is done by comparing T against the sum of $\hat{t}_s$ and $\hat{t}_{us}$. This process is repeated until the next projected reception is from a $\hat{t}_{us}$ interval; the total number of receptions before this happens is equal to m. In more detail, figure 9 shows the algorithm. The algorithm's inputs are T, $\hat{t}_s$, and $\hat{t}_{us}$ and the output is m, the minimum number of zero state values n_a would observe from n_b. In figure 9, T' is the sum of $\hat{t}_s$ and $\hat{t}_{us}$. The algorithm identifies the various possible relationships between T and $\hat{t}_s$ and $\hat{t}_{us}$ to decide how to compute m.

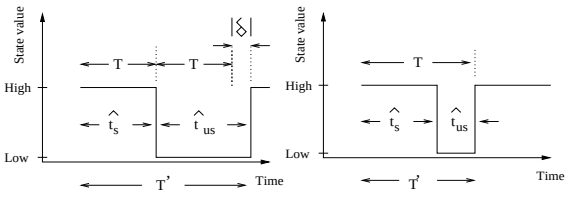
Input: T, $\hat{t}_s$, $\hat{t}_{us}$; Output: m.

0. $T' \leftarrow \hat{t}_s + \hat{t}_{us}$; $m \leftarrow 1$;
1. if($T < T'$ and $T > \hat{t}_s$)
2. $m \leftarrow 2$; //2nd msg in unsaturation
3. else if ($T > T'$) //next msg in a subsequent T' zone
4. $\delta \leftarrow T \% T'$; //modulus
5. while($(\delta - \hat{t}_s) < 0$)
6. $m \leftarrow m + 1$;
7. $\delta \leftarrow (T - (T' - \delta)) \% T'$; //offset from start of next interval
8. else if($T \leq \hat{t}_s$) //next msg in a saturation zone
9. $\delta \leftarrow 0$;
10. do
11. $m \leftarrow m + \frac{\hat{t}_s - \delta}{T} + 1$; //integer division
12. $\delta \leftarrow T(\frac{\hat{t}_s - \delta}{T} + 1) - (T' - \delta)$; //offset from end of current interval
13. while($\delta > 0$);
14. else
15. $m \leftarrow \infty$; //to observe infinite zero-state messages; diagnosis inconclusive

Figure 9: Failure diagnosis algorithm



(a) Condition: $T < T'$ and $T > \hat{t}_s$ (b) Condition: $T > T'$



(c) Condition: $T \leq \hat{t}_{us}$ (d) Condition: $T = T'$

Figure 10: Relationships between T, $\hat{t}_s$, and $\hat{t}_{us}$ of figure 9

First, if $T < T'$ and $T > \hat{t}_s$ (line 1), the next state value of n_b (after an interval of T since the first reception) would be from its $\hat{t}_{us}$ interval and, hence, $m = 2$. Figure 10(a) shows this condition in schematic. Here, the saturated and unsaturated states of n_b are shown as a pulse wave: the "high" value represents the $\hat{t}_s$ interval and "low" the $\hat{t}_{us}$ interval. Along the x-axis is time and the y-axis represents state value. Thus, from the start of the first $\hat{t}_s$ interval (corresponding to the first zero state value n_a received from n_b), the end of a period equal to T (which is when n_a would receive the next message from n_b) falls within the $\hat{t}_{us}$ interval; which is when n_a will receive a non-zero state value. Thus, n_a would receive a non-zero state value on the message after the first zero it received from n_b. Hence, $m = 2$.

Second, if $T > T'$ (line 3), first, it is computed by how much (δ) the next reception is within the next T' of n_b (line 4). Then, this offset is recomputed (line 7) for successive observations (the loop at line 5) until it is greater than $\hat{t}_s$ of the next T' (line 5), which indicates a non-zero state. Until this happens, in each iteration one more zero state value needs to be observed by n_a (line 6). Figure 10(b) shows the schematic.

Third, if $T \leq \hat{t}_s$ (line 8), a similar calculation is done as above. Specifically, in line 12, $T(\frac{\hat{t}_s - \delta}{T} + 1)$ gives the total duration of the consecutive observations made within an offset-adjusted $\hat{t}_s$ interval, and $(T' - \delta)$ gives the effective (offset-adjusted) T' in which the above calculation is done. Thus, the new offset from the end of the current T' is the difference of the above two values (line 12). The process is repeated until the next observation is in the $\hat{t}_{us}$ of the current T' which is when the offset is less than zero (line 13) and indicates a non-zero state. As an example, consider $T = \hat{t}_s$. Then, δ at the end of iteration 1 is (line 12): $\delta = T(\frac{\hat{t}_s - 0}{T} + 1) - (T' - 0)$, $= 2T - T'$. If $2T < T'$, then $\delta < 0$ (line 13 violated) and, thus, the third message after the first zero state that n_a received will be from a $\hat{t}_{us}$ interval of n_b; so, in this example, $m = 3$. Figure 10(c) shows this example in schematic. If $2T > T'$, then line 13 holds. So, in the next iteration, an offset-adjusted $\hat{t}_s$ and T' are used (where the offset is the δ of the previous iteration) to re-compute a new δ.

Lastly, when $T = T'$ (line 14), the next state from n_b always co-

incides with a $\hat{t}_s$ interval (see the schematic of figure 10(d)). Therefore, n_a receives an endless sequence of zero states from n_b and, hence, cannot distinguish between a saturated and a failed n_b.

The above algorithm can be invoked when n_a receives a zero state of a *saturated* n_b. However, assuming n_a has the correct estimates of T, $\hat{t}_s$, and $\hat{t}_{us}$, it will eventually receive a non-zero state of n_b on the m^{th} instance since the first zero. This is when n_b has become unsaturated. So, n_a would not signal a failure which is the correct diagnosis. Hence, the diagnosis has no false positive. Alternatively, if n_b had indeed failed when n_a received the first zero, it would continue to be failed indefinitely. Therefore, n_a would eventually receive more than m zero state values from n_b and then signal a failure — the correct diagnosis (no false negatives).

6.2 Empirical Evaluation

The algorithm of figure 9 is evaluated empirically using the same experimental setup as in section 5. In a given network, one node (n_a) is selected as the monitoring node and then, in turn, all other nodes are selected to fail (n_b), each in a differect simulation with all parameters the same as before. Selecting one node to fail does not limit our conclusions about our diagnosis algorithm which is theoretically capable of detecting arbitrary number of failures. However, multiple failures can cause a practical difficulty of state information not reaching the monitoring node due to a disconnected network. Now, for every other node, n_a estimates T and the simulator estimates $\hat{t}_s$ and $\hat{t}_{us}$, which are fed to n_a using which, it can invoke the algorithm of figure 9 to compute m as described before. In these experiments, node 8 in figure 3(a) and node 13 in figure 3(b) are used as the monitoring nodes. With a different node (the monitored node) failing in each experiment, the average values of m and T over the monitored nodes that are at a given distance from n_a, are computed. Hence, a summary assessment of how many consecutive observations n_a requires to diagnose failure at a given distance is obtained.

To this end, tables 1 and 2 show the average estimates of m, T, and the delay between the time a node fails and n_a actually making a diagnosis for figure 3(a) and 3(b), respectively. *In each experiment, n_a has diagnosed only after the chosen n_b has failed. Thus, the properties of no false positives and no false negatives of the diagnosis algorithm are corroborated by these empirical results.* Thus, not only does e-PTC yield highly effective adaptive response in distributed dynamic environments, but also using the information exchanged by e-PTC, accurate state diagnosis can be achieved.

Note the diagnosis delay increases with increasing distance of the failures because of the increasing delay (T) in receiving information from distant nodes. The value of T is determined by the topology, load, call pattern, and the information-sharing protocol used. By reducing T, such as by distributing information more frequently and along multiple paths, the detection delay can be reduced; this is part of our future work. The delay is also due to using a single monitoring node. By choosing multiple such nodes, the maximum distance of any node and at least one monitoring node can be restricted such that the delay is within tolerable limits. Nevertheless, we verified the theoretical properties of the diagnosis algorithm using a single monitoring node which actually provides a "harder" test than using multiple ones.

Table 1: Diagnosis performance on the topology of figure 3(a)

Min dist of target	T		m		Detection delay
	avg	stdev	avg	stdev	
1	420	50.7	2.8	1.2	756
2	1559	562.4	2.9	1.4	2962.1
3	7402	5339.3	2.3	1.2	9622.6
4	28616	15895.0	2	0	28616

Table 2: Diagnosis performance on the topology of figure 3(b)

Min dist of target	T		m		Detection delay
	avg	stdev	avg	stdev	
1	372	11.2	2.2	0.3	446.4
2	4148	491.1	2.0	0.1	4148.0
3	10807	658.9	2.0	0.2	10807.0
4	20265	1870.5	2.1	0.4	22291.5
5	33336	4668.6	2.5	1.3	50004.0
6	44569	15631.4	2.0	0.0	44569.0
7	63416	8510.4	2.1	0.6	69757.6
8	132917	15006.6	1.85	0.3	112979.4
9	464743	13766.6	2.0	0.1	464743.0
10	425047	21690.3	2.0	0.1	425047.0

7. CONCLUSIONS AND FUTURE WORK

In this paper, we develop a new information-sharing algorithm, e-PTC, for distributed resource allocation problems that can generate highly effective adaptive behaviour in dynamic environments and achieve distributed diagnosis of state changes. Empirical evaluation of e-PTC in a simulated call routing application confirms its advantages over the previous state-of-the-art in this area. Specifically, with dynamic state changes caused by network failures, e-PTC achieves up to 20% higher call throughput, up to 3.5 times faster recovery of throughput after failures, and provides a novel algorithm for diagnosing failures without false positives and false negatives. These improvements are earned by e-PTC at the cost of sending more messages than PTC. However, to address this, we plan to investigate ways of limiting the message overhead of e-PTC by using selective communication in which the agents can use a notion of information redundancy to decide when to communicate.

8. REFERENCES

[1] E. H. Durfee and V. R. Lesser. Partial Global Planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, 1991.

[2] P. S. Dutta. *Adaptive Distributed Resource Allocation Using Cooperative Information-Sharing*. PhD thesis, University of Southampton, 2005.

[3] P. S. Dutta, N. R. Jennings, and L. Moreau. Cooperative information sharing to improve distributed learning in multi-agent systems. *JAIR*, 24:407–463, 2005.

[4] Floodnet: Pervasive computing in the environment, 2004. http://envisense.org/floodnet/floodnet.htm.

[5] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaugmann, 2003.

[6] P. J. Gmytrasiewicz and E. H. Durfee. Rational communication in multi-agent environments. *Autonomous Agents and Multi-Agent Systems*, 2001.

[7] C. V. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of AAMAS 2003*, pages 137–144, 2003.

[8] M. Lemaitre, G. Verfaillie, H. Fargier, J. Lang, N. Bataille, and J. M. Lachiver. Equitable allocation of earth observing satellites resources. In *Proc. of the 5th ONERA-DLR Aerosapce Symposium*, 2003.

[9] N. Lynch. *Distributed Algorithms*. Morgan Kauffman, 1996.

[10] V. Tamarapalli and S. Srinivasan. Survivability in WDM networks under multiple link failures. In *Proceeding of Optical Communications Systems and Networks*, 2004.

[11] C. J. C. H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8:279–292, 1992.