Learning Node Characteristics for Reliable Leader Election

Isha Tarte * Rochan Avlur *
University of Texas at Austin
{tarteisha, rochan.avlur}@utexas.edu

Abstract

Consensus algorithms are vital to building fault-tolerant distributed system. As multiple nodes work together on a single task, a reliable leader is essential to organize the task distribution and coordinate the decisions. However, scenarios such as resource allocation imbalance and different node failure-rate can lead to sub-optimal leader election. One can overcome this by utilizing underlying node characteristics (e.g., failure rate and processing capabilities). We propose a leader election algorithm which balances between random (*exploration*) and greedy (*exploitation*) approaches, leading to fewer faulty nodes being elected as leader; thereby minimizing response delay and downtime due to election rounds. To address adaptive control of exploration-exploitation, we design *Bandit Leader Election* (BLE), a simple but effective protocol for bidirectional connected networks. BLE solves this tradeoff by cooperatives failure-rate estimation in a multi-agent multi-arm bandit setting. We evaluate the proposed method in environments with stationary and non-stationary node failure-rate, demonstrating its effectiveness and reduction in election rounds. We show BLE outperforms deterministic & randomized election algorithms for stationary environments. Our code is available at https://github.com/isha97/bandit-leader-election.

1 Introduction

Building a reliable system in the presence of faulty nodes requires coordinating processes to reach consensus. Consensus can be achieved by selecting one node among the pool as the leader who is responsible for coordination. Besides coordination, the leader also controls various activities like task allocation, result aggregation, efficient resource sharing, clock synchronization, and communication among the nodes of the system [5]. It is essential that there is always an active leader node in the system to perform all the responsibilities.

Leader election has been studied extensively over the years in various distributed computing models starting with the standard CONGEST model of networks [12]. In existing studies of leader election algorithms, randomized and deterministic methods dominate as they make very few to no assumptions regarding the underlying characteristics of nodes in a distributed system [7, 20]. All these algorithms relies on the current state of the node to decide upon if it will be a good leader. They don't consider the inherent properties of different nodes such as failure probabilities, resource availability which can defer if the cluster is made up of heterogeneous machines differing in processing capability. Futhermore, most of the leader election algorithms aims to minimize the number of message exchange in a leader election round and lesser emphasis is given to reduce the frequency of leader elections that happens within the system in a given time frame. A leader failure causes a client to wait for a longer time than normal to get the response back. There would be an inherent delay of timeouts set in various algorithms such as [18], [7] before a new leader can be selected.

Motivating scenario 1. Failure is more the norm than an exception in large-scale commercial datacenters of today [1]. There could be multiple reason for a node to go down like a storage node or networking switch can be overloaded; a node binary or operating system may crash or restart; a machine may experience a hardware error; automated repair processes may temporarily remove disks or machines; or a sub-cluster could be brought down for maintenance or deployment [9]. From prior works analysing failure causes, nodes are known to exhibit *stationary* or *non-stationary failure-rates* [4, 9]. For instance, nodes with ageing components may be more susceptible to failure due to long-term mechanical damage or nearing their end-of-life. On the other hand recently upgraded nodes are more reliable, lowering their failure-rate. Estimating this stationary distribution of failure-rate for individual nodes can be exploited to select a leader which has much lesser probability of failure.

*Authors listed in alphabetical order.

Motivating scenario 2. Nodes may also exhibit varying *performance characteristics*. For example, Microsoft operates Cosmos, which involves more than 20 hardware generations of machines (with varying CPU cores, RAM, HDD/SSD) from various manufacturers and software configurations (e.g., mapping of drives to SSDs/HDDs) [26]. We assume system information is not shared among nodes as otherwise, estimating node performance becomes relatively simple. Dynamic network incidents such as network failures or reconfigurations, as well as high control plane load may result in *network overload* on the path to the leader and thus deteriorate the resulting cluster performance. Estimating this distribution of node characteristics can be exploited to select a leader which has highest available resources.

Motivating scenario 3. Consider the nodes of the system running on VMs (Virtual machines) rather than physical machines. VM migration describes the process of moving a VM from a hosting physical server to another destination server at the same or another data center [24]. The majority of cloud management operations are supported by VM migration [24]. Each underlying physical machine has its own stationary failure-rate and performance characteristics while the VM's characteristics depends on the underlying physical machine. Due to live migration, a VM's "failure-rate" undergoes variations as it migrates between physical machines, inducing a non-stationary failure-rate on the VM. In this case the node characteristics can be non-stationary and we experiment with using learning based approach to select a leader in such a scenario.

Our contributions. Estimating the above stationary and non-stationary node characteristics empirically in a distributed system is a challenging task. The intuition is clear however: an exploratory or random approach must be taken when the true node characteristic distributions do not match the empirical estimates. When they are approximately close, we must exploit this information to elect leaders that minimize downtime and maximize reliability. The design challenge is two folds: (i) to quantify when to perform exploration and exploitation, and (ii) empirical estimation of the true node characteristics distribution; via an asynchronous distributed algorithm with fewer overall message transmission. Hence, we formulate the exploration-exploitation trade-off as a multi-agent multi-arm bandit problem. Our main contributions are:

- An adaptive distributed protocol for estimating relevant node characteristics based on an exploration-exploitation tradeoff.
- A leader election algorithm modeled as a *multi-agent multi-arm bandit* problem which uses the estimates of node characteristics to select a leader and arrive at consensus with other nodes.
- We empirically show that learning-based LE outperforms or matches randomized LE and deterministic LE across different settings under certain assumptions.

2 Related Work

Castro *et al.* [7] solves the problem of leader election in a system with byzantine faults where the leader of each round is pre determined and given by *view_number % N* where *N* is total number of nodes in the system. [18] uses a heart-beat based mechanism to trigger leader election that is done by voting. The candidate sends *RequestVotes RPC* to all other nodes and the winner is decided by the one which receives maximum number of votes. A different approach to solve the problem of leader election especially in a ring network is the use of probabilistic algorithms [10]. [20] provides a Randomized leader election algorithm based on a balls and bins abstraction and works in two phases. In the first phase, the algorithm reduces the number of contending nodes and in the second phase, it resolves a winner. [12] also propose a randomized leader election algorithm in complete networks where candidates attempt to progress towards becoming the leader based on the Rank.

There have been prior work in the literature that consider node characteristics such as performance and failure-rate over a finite time window for leader election [5, 21]. [21] selects a leader which minimizes the average cluster response time and is built on top of raft. Resource contention, network dynamics, and heterogeneous resource allocation to nodes hosting the replicas can affect the performance of a node and a leader is selected based on these metrics. However, it uses a logically centralized leader selector called SEER which each replicas send a heartbeat to. SEER is responsible for selecting the leader after receiving the metrics from each of the nodes. In our work, we want to use the metrics to achieve consensus without relying on a single node for leader election.

[19] proposed a leader election algorithm for a mobile adhoc network which is based on the group membership detection algorithm. Every process detects a member process in the group in a synchronous distributed system and a new leader is elected depending on the process weights belonging to the set of detected processes. [5] propose an algorithm for bidirectional ring networks to elects a node with a minimum failure rate and load as the leader. It uses a leader coefficient which is a linear combination of resource utilization and failure rate to select the leader node. However, the authors don't discuss on how are these metrics shared among the nodes and estimates of failure rate are maintained. Also, we aim to use learning based leader election on a fully connected network instead of a ring network.

[17] propose a system which uses previous workload and observed latencies to reconfigures the leader set in scenario with imbalanced workloads. [25] obstructs Byzantine servers from being elected in leader election by imposing hash compu-

tation on new election campaigns, leveraging Proof-of-Work and Raft. In [8], clients optimize the selection of their leader replicas in a BFT setting, by sending special probe messages used to collect end-to-end response times. In contrast, we focus on non-byzantine setting and designing a single-leader election algorithm.

[11] studies distributed exploration in Multi-Armed Bandits where k players collaborate in order to identify an ε-optimal arm. It demonstrate a non-trivial trade-off between the number of arm pulls required by each of the players, and the amount of communication between them. [22] models the online task assignment problem in distributed wireless computing as a contextual combinatorial bandit. In our work, we aim to obtain closest estimate for node characteristics even if it causes more number of message exchanges.

3 System Model

We consider a distributed system composed of N nodes, where, N is an integer and $N \geq 3$. Formally, this system can be represented by an undirected graph $G = (V, E)$ where V is the set of nodes ($|V| = N$ and $V = \{i\}$ where $i = \{1, 2, \ldots, N\}$), and E is the set of network connectivity between the nodes ($E = \{e_{ij}\}$, where, e_{ij} is the edge between nodes i and j; $i \neq j$ and $i, j \in V$) [6]. We assume that the graph G is fully connected, i.e., there exists a edge between every pair of nodes i.e. $\exists\, e_{ij} \in E\ \forall i, j \in V$.

3.1 Assumptions

Our system implementation make the following assumptions about the nodes, network and node counts. These are similar to the assumptions made by other consensus algorithms such as [13, 16].

Nodes:

- Nodes may experience failures but the failures are fail-stop i.e. the nodes stop responding to any messages or requests once it fails.
- Nodes may re-join the protocol after failures (following a crash-recovery failure model).
- The nodes do not collude, lie, or otherwise attempt to subvert the protocol.
- We don't rely on any failure detector i.e. it's up to individual nodes to detect that the leader has failed by means of timeouts.
- Nodes are not aware of the type, failure rates and performance characteristics of other node before hand.

Network:

- Every node can send messages to every other node.
- Messages are sent asynchronously and may take arbitrarily long to deliver.
- Messages may be lost, reordered, or duplicated.
- Messages are delivered without corruption (There is no byzantine behaviour.)

Number of Nodes: We consider a configuration of N nodes where $|N| = 2f + 1$. Here f denotes the maximum number of nodes which can be failed at a time. We derive this from Paxos [13] which requires $2f + 1$ acceptors, always leaving at least $f + 1$ acceptors to maintain the fault-tolerant memory. In general, a consensus algorithm can make progress using $|N| = 2f + 1$ processors, despite the simultaneous failure of any f processors[14].

In our work, we only concern with leader election and don't discuss about how nodes agree on the order of updates and eventually commit the update requests in the replicated log. We show that the learning based leader election proposed provides liveness i.e. the system is able to make progress even in case the primary fails.

Definition 1. *(Failure-Rate) $F^i(t)$ is a random variable representing the failure rate of a node i. It is the probability with which node i fails at time t.*

4 Distributed Multi-Agent MAB

Consider a distributed multi-agent multi-armed bandit (MAB) problem in which M agents make sequential choices among the same set of N arms with the goal of maximizing their individual reward [15]. In our case, agents and arms are nodes which cooperate by indirectly sharing their estimates of node characteristics of each arm over the network. Let $\mathcal{T} = \{1, 2, \ldots, T\}$ denote a sequence of decision epochs faced by all non-faulty nodes. Using its local information, each node $i \in \{1, \ldots, M\}$ selects an arm (i.e., node) at epoch $t \in \mathcal{T}$.

We study two models which determine the dynamics of the estimates maintained by each node. We first discuss the *stationary* model and in the section following it, discuss the more complex *non-stationary* model. The objective of the distributed cooperative multi-agent MAB problem, which is shared by both models, is to maximize the expected cumulative group reward. In our setting the reward is maximized by selecting an leader which has the least probability of being failed. In other words, our estimates represent the cost of selecting the arm and in MAB, the task is to minimize the cost.

4.1 Stationary Failure Model

In this model, we assume node characteristics (i.e., failure probability) remains the same over T. This problem can be modelled by assuming that the nodes are physical machines

and their underlying properties do not change. We use bandit-based algorithms to select candidate leader nodes. This involves a trade-off between exploration and exploitation. Each node maintains a node count vector denoted by n_i^j for node j which is the number of times a node j's estimate is updated. Exploration refers to selecting a node $j \in \arg\min_j n_i^j$ while exploitation refers to selecting the node with least failure probability. The two algorithms which we used are:

- **ε-greedy**: Has a hyperparameter ε which controls the exploration. The algorithm selects a random node with ε probability and selects node with the lowest failure probability otherwise. As time progress, ε is decayed by a factor to reduce the exploration as the estimates become more accuracte.

- **UCB:** Rather than performing exploration by selecting an arbitrary arm from all arms with uniform probability, UCB algorithm changes the exploration-exploitation balance as it gathers more knowledge of the environment [2]. The node selected as the candidate leader is given by

$$i = \arg\min_{i \in N} F^i - c * \sqrt{\frac{\log t}{n^i}} \tag{1}$$

  Here, c is a constant hyperparameter that controls weight given to exploration and t is the current time step. This algorithm selects the node with the lowest failure probability and weighted exploration amount (i.e., a node less explored is more likely to be selected).

4.2 Non-Stationary Failure Model

In the more complex *non-stationary model*, the underlying distribution of failure probability of each node can change an arbitrary number of times [3]. We assume a variation budget V constraints the stochasticity in the non-stationary environment. While limiting the possible evolution in the environment, it allows for numerous forms in which the expected rewards may change: continuously, in discrete shocks, and of a changing rate.

A simple extension to both ε-greedy and UCB can help in balancing exploration and exploitation in non-stochastic environment. Specifically, a fixed length memory buffer is used to retain that latest returns going back a fixed length. To take decisions, the decision agent only consider values in this buffer to update its own estimates. This ensures that very old values are not used to build the estimates.

5 Architecture

The main components of our system are:

1. **Client:** Sends request messages to leader node and expects a reply.

2. **Nodes:** Responsible for processing requests. As we are only concerned with leader election, nodes are responsible for ensuring an active leader is present at all times.

3. **Environment:** As our system uses the underlying characteristic of all the nodes such as failure probabilities or resource availability, an independent component is required that is aware of these properties and selectively make nodes aware of their characteristics (e.g., letting a node know when it fails).

5.1 Request-Reply Flow

Client sends *client_request* messages represented as <*request_id, timestamp*> to the node it believes is the current leader. It expects a reply within certain time. In a normal scenario, the client receives a reply from the leader. The reply message is of the form <*request_id, leader_node, timestamp*>. But if the client does not receive the reply within the expected time, it broadcasts the request to all the nodes of the cluster.

5.2 Bandit Leader Election

The leader is responsible for replying to the client requests. It also sends a *request_broadcast* message (<*request_id, timestamp*>) to all the nodes upon receiving the request from the client. We now explain what can cause a new leader to be elected. When a non-leader node receives a *client_request* message, it waits for a certain time and checks if it has received the *request_broadcast* message for the *request_id* from the current leader. If not, it assumes the leader is failed.

The following phases are part of a leader election round:

1. **Candidate selection.** Each node maintains the estimate of failure probability (or performance characteristics) of all the nodes and uses the algorithm described in the previous section to decide upon the candidate leader node(s). It then broadcast the *share_candidate* message (<*sender_id, candidate_nodes, timestamp*>) to all other nodes. Figure 1a shows how the *share_candidate* messages are exchanged.

2. **Leader selection.** Each node waits for f *share_candidate* messages and then selects the leader based on the voting mechanism. This ensures a consensus of total $f+1$ nodes in selecting a leader. It selects the node which is present in *candidate_nodes* of maximum number of nodes as the leader. In case of ties, candidate with a smaller index is selected.

3. **Leader election confirmation.** The node which is selected as the leader sends *confirm_election* (<*leader_node, timestamp*>) message to all the nodes. The other nodes wait for this message and then update the leader. If nodes do not receive a *confirm_election*

message within a certain time, next round of leader election is started. Figure 1b shows how *confirm_election* messages are broadcast.

5.3 Estimating node characteristics

Each node $i \in N$ maintains an estimate of characteristics of all nodes in the system. Failure probability is one such characteristic of a node and denote its estimated value for node j by $\widehat{F_i^j}$. Each node initially starts with an estimate for all the other nodes drawn from a normal distributions (see Sec 6.3). A node i increases $\widehat{F_i^j}$ if it expects a reply from node j but does not receive it. On the other hand, it decreases $\widehat{F_i^j}$ if it receives an expected response from node j.

Ping Message. Each node sends a *ping* message represented by <*sender, timestamp*> to other nodes to check their health. In our simulation, we set the delay between sending *ping* messages to be much greater than incoming client request rate. Also, a node waits for a *ping_reply* before sending another *ping* message to the same or different node. Node to be pinged is decided based on an exploration algorithm which selects the node i given by,

$$i = \arg\max_{j \in N} \sqrt{\frac{\log \mathcal{T}}{n_i^j}} \tag{2}$$

Here, $\mathcal{T}$ is total number of times exploration takes place. The above equation ensures that node i that is selected is of the node that has been explored the least amongst all nodes.

Increasing node estimate. Estimate of node j as maintained by node i is updated as,

$$\widehat{F_i^j(t)} = \frac{\widehat{F_i^j(t-1)} * n_i^j(t-1) + 1}{n_i^j(t-1) + 1}$$
$$n_i^j(t) = n_i^j(t-1) + 1$$

The following events can cause an estimate to be increased:

1. When node i sends a *ping* message to node j and does not get a reply back from j within a timeout limit ($\widehat{F_i^j}$ is increased).
2. The node i increase it's own failure estimate upon recovery from failure by comparing the current time and the last message timestamp ($\widehat{F_i^i}$ is increased).
3. The current leader's estimate is increased across all non-faulty nodes when the leader is assumed to be faulty (i.e., the node receives a *client_request* message but not the *request_broadcast* message). $\widehat{F_i^j}$ is increased where leader is node j and $\forall$ i not faulty.

Decreasing node estimate. Estimate of node j as maintained by node i is decreased as,

$$\widehat{F_i^j(t)} = \frac{\widehat{F_i^j(t-1)} * n_i^j(t-1)}{n_i^j(t-1) + 1}$$
$$n_i^j(t) = n_i^j(t-1) + 1$$

The following events can cause the estimate of node j to be decreased:

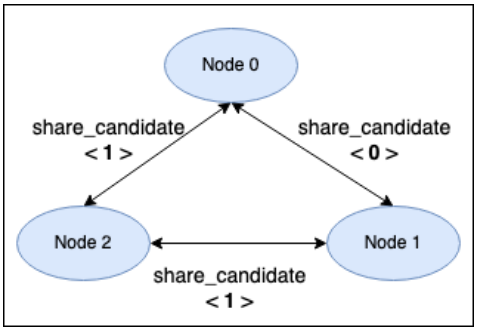
1. When a node i sends a *ping* message to node j and get back the reply from j within the timeout.
2. If j is the leader and node i receives a *request_broadcast* message from j.
3. If i is the leader and i receives a reply for *request_broadcast* message from j.
4. If j is the selected as the leader and i receives a *confirm_election* message from j.
5. If i receives a *share_candidate* message from node j.

Our aim was to minimize message exchanges in BLE protocol while enabling estimation of node characteristics by leveraging existing messages like *request_broadcast*, *confirm_election* and *reply_broadcast*.

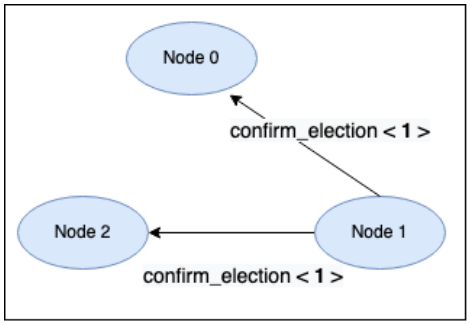
Penalizing faulty leaders. If node i is currently the leader and node j detects that i is not responding to client's request (by timeouts), j penalizes $\widehat{F_j^i}$ by increasing its value temporarily by a large constant value. This prevent node j from selecting node i as the candidate leader node again. The failure probability is reduced again when j receives the reply of a ping message from i confirming that it has recovered.

6 Experiments

In our experiments, the key question we seek to address is whether trade-off between exploratory and greedy selection can be controlled by cooperative estimation of node characteristics. In this context, we detail experiments performed to investigate whether BLE leads to improved performance across varying environment settings. In particular, our experiments focus on two types of environments: *stationary* and *non-stationary*. First, we investigate performance of BLE in stationary setting where node characteristics take the form of failure probabilities fixed at a constant value. Then, we investigate performance in non-stationary setting with node characteristics are modeled by resource availability and can change over time.



(a) Sharing Leader Candidates



(b) Leader broadcasting confirm election

Figure 1: Steps in Bandit Leader Election Algorithm

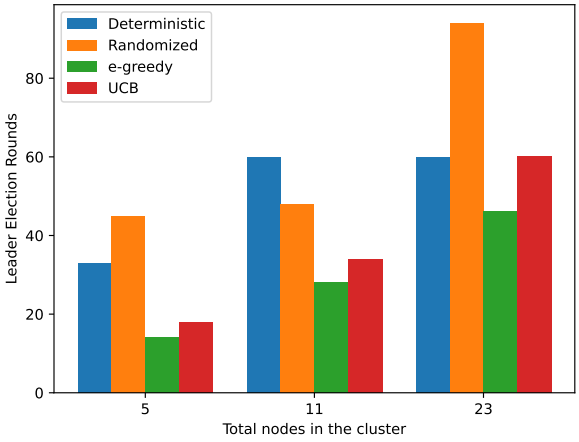


Figure 2: Number of leader election rounds for baseline and BLE algorithms with 2500 requests with varying node population.

Machine Type	Count	Failure Prob.
I	463	0.95
II	2025	0.15
III	1114	0.7
IV	717	0.2
V	810	0.8

Table 1: Distribution of machine types with their respective failure probability (see [4] for further details).

6.1 Setup

In need of fine-grained control of environmental parameters for fair comparison and due to resource constraints, we simulated various baseline LE and BLE algorithms. In our implementation, nodes and clients are simulated as processes, all running in parallel on a multi-processor machine. Communication among nodes and between leader & client was implemented using TCP server-client protocol. For persistent storage, local machine file system was used. In addition to node and client processes, an environment process, responsible for updating node failure probabilities and for logging was launched in parallel. All baseline and BLE algorithms were implemented to work within this environment setup. For reproducibility, all random functions used in the simulation are parameterized by input seed.

Baselines methods. As baselines, we compare BLE against two popular LE algorithms: *deterministic* (DLE) and *randomized* (RLE). For DLE, we followed the algorithm detailed by Castro *et al.* [7]. Here, nodes are identified by an integer ID. Upon leader failure, identified by timeout, the node with the next lowest ID elects itself as the leader and broadcasts an update message. If no election confirmation is received within a timeout, the next lowest ID elects itself and so on. Hence, ID of leader node follows a cyclic order of values $\{0, 1, 2, \ldots, N\}$. In RLE, we implement a simple randomized election algorithm where every node selects a *candidate* leader from the node pool such that all nodes have equal probability of being picked as a candidate. All nodes broadcast their candidate list and the candidate with maximum votes gets selected as the leader. Ties are broken by selecting the node with lower index.

Broadcasting messages. Many efficient message protocols for randomized election have been proposed [20]. While empirical analysis of message transmission is a component of analyzing LE algorithms, we opt to simplify our implementation and use broadcast messaging with all algorithms. Independently, we perform message complexity analysis of our algorithm, results of which can be found in Sec. 7.1.

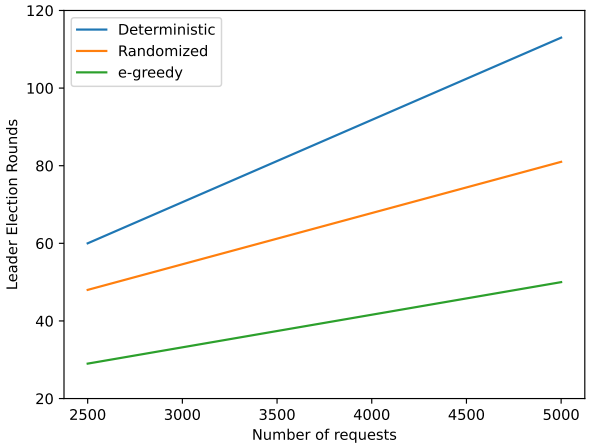


Figure 3: Number of LE rounds with different number of client requests for system with 5 nodes.

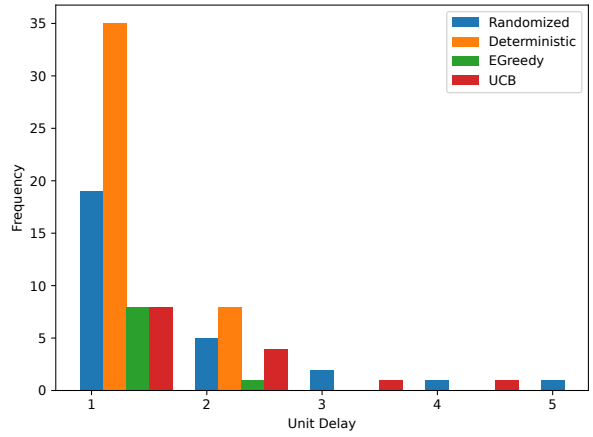


Figure 4: LE time-elapsed frequency

6.2 Simulating Node Failure

To replicate real-world node failure rate and system behaviour, we take statistics from prior work on large-scale system reliability analysis. In particular, we use Birke *et al.* [4] work on failure analysis of among virtual and physical machines. For the purpose of simulating system behaviour and in the interest of time, we replicate one week of node failures in 30 seconds of simulation time. Accordingly, we scale all other node failure properties. Table 1 illustrates distribution of machine types with their respective failure probability. To study relation between total nodes and LE performance, we repeat experiments changing the total number of nodes in the system (5, 11 and 23 nodes). Node type is drawn from from this distribution with probability of selecting a node being proportional to the node count depicted in Table 1. To analyse worst-case performance, we bound the minimum number of failed nodes as a fraction of the maximum failed nodes possible in the system (we use 0.8 for all experiments). That is, given our non-Byzantine system, number of failed nodes at time t is bounded by $(0.8) \cdot f \leq f_t \leq f$.

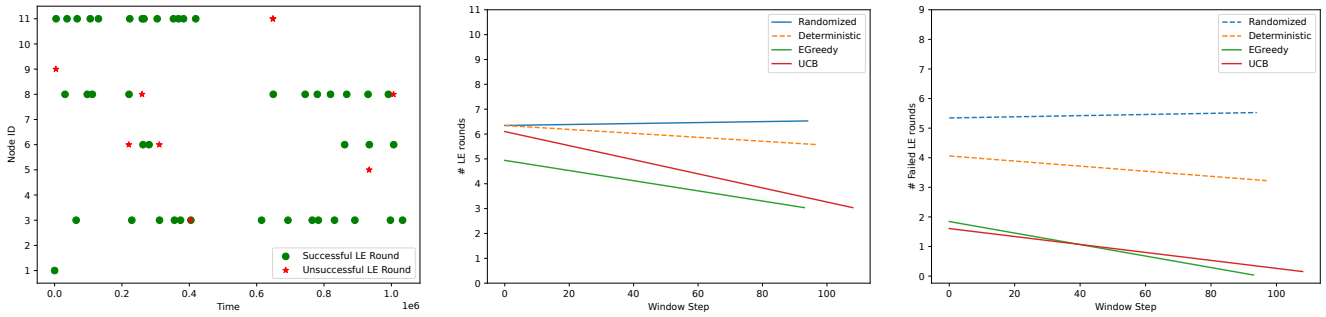
Repair duration, the time it take for a node to recover from failure, is another key parameter adjusted by borrowing from [4]. The authors show that distribution of node repair duration follows a *Log-normal* distribution with parameters $(\mu = 1.80, \sigma^2 = 2.07)$. We scale repair duration according to failure rate. To study how LE performance scales with time, we perform experiments for 2500 and 5000 client requests. Time between two consecutive requests is set to 2 seconds.

6.3 Stationary Node Characteristics

# of leader election rounds. We compare number of LE rounds when using baseline and BLE algorithm. In Figure 2, we illustrate the number of LE rounds for a system with 5, 11, and 23 nodes, receiving 2500 client requests. Irrespective of number of nodes, as expected, BLE algorithm reduces total number of LE rounds. Baseline algorithms tend to perform similarly although randomized LE performs poorly with 23 nodes. Increasing total nodes in the system increased the number of LE rounds for the same duration. We believe this is due to increase in failed nodes. Interestingly, amongst the BLE algorithms, e-greedy-LE slightly outperforms UCB-LE. We attribute this difference to the exploration weight hyperparameter, c, in UCB and believe that when tuned over multiple trials, will outperform e-greedy-LE.

Varying client requests. Next, we compare performance of LE across different time durations. As number of requests is proportional to total time the system is serving to queries, we perform LE with 2500 and 5000 client queries. We expect the number of LEs to increase at the same rate at which number of requests increase (i.e., slope should be close to 1). Figure 3 illustrates number of LE rounds with increasing number of requests. Baseline LE algorithms have a steeper slope while BLE algorithms have a milder slope. In the latter case, exploration performed within the first 2500 client requests learn good estimates of node failure probability, hence reducing the number of LE rounds during the next set of 2500 requests (i.e., only 19 LE rounds occur in the last 2500 requests compared to 31 in the first 2500 requests for BLE). From this result, we can infer that running the system for 2x duration almost doubles the LE rounds for baseline LE algorithms while BLE results in only a $\sim$ 1.6x increase.

Leader election frequency. To highlight the fact that estimating failure-rate leads to fewer LE rounds as time progresses, we plot frequency of LEs within a fixed sliding time-window as shown in Figure 5b. First we compute the frequency for each sliding-window step and then perform linear



(a) LE Rounds (b) Successful LE (c) Unsuccessful LE

Figure 5: Frequency of LE rounds over time with 5000 client requests and 11 nodes.

interpolation to plot the trend. We observe both BLE algorithms display negative slopes, indicating a reduction in LE rounds as time progresses. Randomized LE displays a positive slope while deterministic LE displays a nearly horizontal slope (we think random shows a positive slope due to stochasticity in random election. On average, both should have a constant value). Likewise, we plot frequency of recurring leader failure (i.e., when a leader fails and the elected candidate node is also failed) within the same fixed sliding time-window shown in Figure 5. Ideally, we want recurrent leader failures to reduce over time as nodes with lower failure probabilities are more likely to be elected as leader. Our observations match our expectations, with BLE displaying a steeper negative slope while baseline algorithms displaying either a gentle negative or positive slope.

LE round duration. As we simulate LE algorithms, we cannot rely on wall-clock times to compare the time elapsed in a LE round. Rather, we measure time elapsed in terms of unit LE rounds and plot a frequency graph of recurring LE rounds as shown in Figure 4. Here, the x-axis represents unit LE rounds and y-axis represents the frequency of occurrence. Ideally, a good LE algorithm should have very low total counts across all unit time elapsed and should describe a heavy left-tail distribution. This corresponds to fewer recurring LE rounds overall and when they do occur, the number of rounds are very few (i.e., a good replacement leader is elected quickly). We observe that BLE algorithms perform well in this setting, displaying characteristics that we expect from good LE algorithms. It outperforms both baseline algorithms suffering from greater delay frequency.

Estimating node characteristics. The objective of BLE is to estimate the node characteristics (i.e., failure probability) in a cooperative and distributed manner. To validate our protocol closely estimates the true underlying failure probability of nodes, we plot them in Figure 7. Solid line represents estimated value while dashed line represents true value. During node initialization, estimates are set to random values drawn from a Gaussian distribution representing a prior belief about the node failure (in all our experiments, we draw from $\mathcal{N}(0.2, 0.1)$). We observe that BLE protocol is able to learn good estimates of node failure rate by balancing exploration and exploitation. Figure 8 shows that all the nodes get selected as the leader initially but as time progress the nodes with least failure probabilities (i.e., 1 and 2 only gets selected as the leader).

6.4 Non-stationary Node Characteristics

In non-stationary setting, node characteristics can change over time owing to multiple reasons as described before. In our simulation, we assume a heterogeneous pool of machines with varied cores and memory count derived from [26] to model the environment. Zhu *et al.* [26] publish distribution of CPU usage for machines with different hardware generations. We attribute higher CPU utilization to higher probability of not responding to requests. We simulate VM migration by rotating the nodes among the heterogeneous pool of physical machine at fixed intervals. Figure 6 illustrates the number of LE rounds for each algorithm. We observed using BLE results in fewer LE rounds in comparison to baseline algorithms. However, the improvement is not significant when compared to the stationary model discussed before in Sec. 6.3. Interestingly, we observed that egreedy-LE marginally outperforms egreedy-LE(B) (i.e., egreedy-LE with buffer). In the latter case, there exists a negative correlation between buffer size and node characteristics variation frequency. By tuning the buffer size to an optimal value, more performance can be extracted. We attribute the marginally lower performance of egreedy-LE(B) to this particular issue.

7 Analysis

In this section, we perform analysis of BLE from a theoretical persepctive and describe its results.

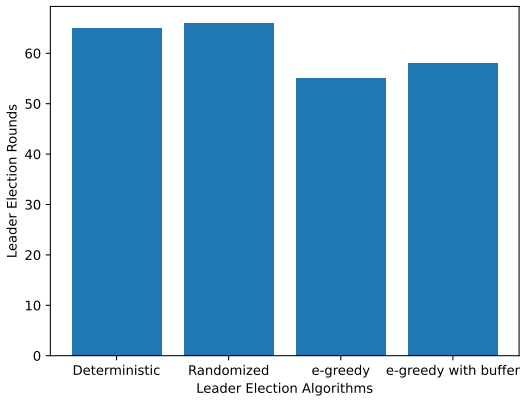


Figure 6: LE Rounds with different algorithms for Non-stationary environment

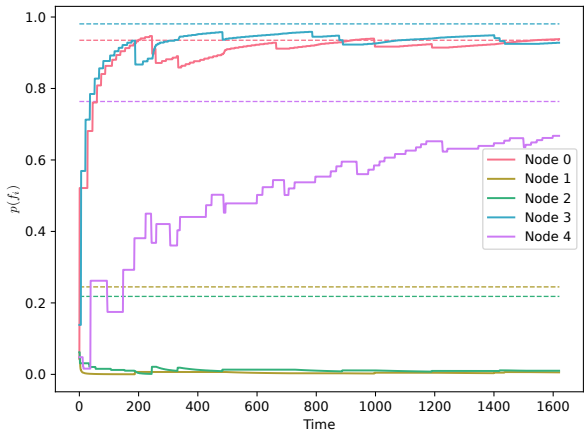


Figure 7: Node failures estimates after 2500 requests.

7.1 Message Complexity

Sec. 5.2 explains the three steps in an election round of BLE. As the *share_candidate* message is sent by broadcast to all nodes, the messages exchanged are of the order $O(N^2)$. The leader broadcasts *confirm_election* message to all nodes which add an additional $O(N)$ messages. Therefore, in total, there are $O(N^2+N)$ messages exchanged in a single LE round. If we fix a time window T and assume BLE causes M LE rounds, then the total messages exchanged in that time window, only counting those part of LE, is $m_l = O(M(N^2+N))$.

Baseline LE algorithms will at least involve a *confirm_election* round and therefore, lead to at least $O(N)$ message exchanges in a single LE round. If there are X LE rounds during time window T, in total there will be $m_b = O(X \cdot N)$ messages exchanged. As our experiments in Sec 6.3 showed that $M << X$, let us assume that $X = kM$ for some $k > 1$. Then, the following lemma holds.

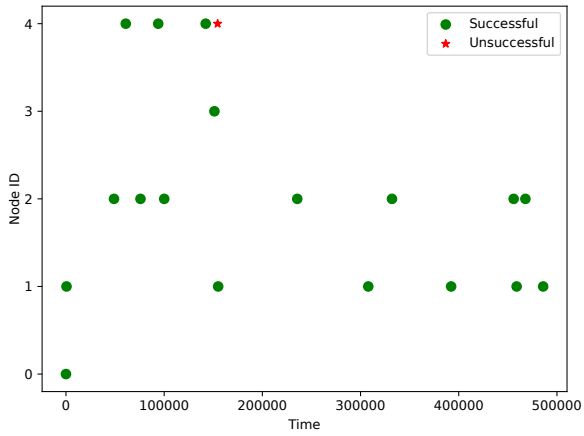


Figure 8: Node selected as the leader for 2500 requests.

Lemma 7.1. *Given a fail-stop model consisting of $N = 2f+1$ nodes with at most f nodes faulty at any time. Let m_l and m_b represent the number of messages exchanged using BLE and baseline LE algorithms respectively as described above. Then, the following inequality holds,*

$$m_l \leq m_b \quad for, \; k \geq N+1. \tag{3}$$

Proof. We have

$$m_l - m_b = M(N^2+N) - X \cdot N$$

Give that, $X = kM$ and $k > 1$,

$$\begin{aligned} m_l - m_b &= M(N^2+N) - kM \cdot N \\ &= N \cdot M(N+1-k) \\ &\leq M \cdot N(N+1-(N+1)) \\ &\leq 0 \\ &\implies m_l \leq m_b, \text{ if } k \geq N+1 \end{aligned}$$

□

As we showed in 6.3, over more requests, m_b increases by a larger proportion than m_l. Moreover, for smaller values of N and a large window size T, we can have $k \geq N+1$ and hence, reduce the total message exchanges by using BLE algorithm.

7.2 Liveness

To provide liveness, nodes must elect a new leader if the client's requests are not served. Two key design decisions of BLE help achieve this. First, to avoid starting LE too soon, a node waits for certain time after it receives *client_request* to start broadcasting the *share_candidate* message. As there are atleast $f+1$ nodes which are running, the node will receive f *share_candidate* messages and then wait for a certain time and select the leader using majority voting. If it doesn't receive f

share_candidate messages within certain time, it starts a new round of LE.

Second, if a node receives a set of valid *share_candidate* messages from other replicas for timestamp greater than its current timestamp, it assumes a leader election round is in progress and sends a *confirm_election* message if it's supposed to be the leader, even if it's timer for *request_broadcast* has not expired. This prevents it from starting the next state change too late.

8 Conclusion & Future Work

Consensus algorithms and in particular, leader election, are vital to building fault-tolerant distributed system. Conventional leader election algorithms either predominantly perform random (exploration) or greedy (exploitation) selection of nodes during leader election. Meanwhile, prior methods show estimating node characteristics provides a way to elect more reliable leaders. In this project, we explored a way for nodes in a system to automatically estimate relevant node characteristics for electing a reliable leader. We formulated the problem in a bandit framework and proposed a method for adaptive control of exploration and exploitation selection. We empirically show that our method displays better or similar performance to random and deterministic leader election.

There are several open future directions to extend BLE algorithm. In our design, nodes broadcast their estimates to all other nodes in the system. However, a more efficient messaging protocol can be used where estimates are shared only amongst a smaller subset of nodes in the system [15]. An interesting future direction would be to use more efficient bandit variants such as contextual bandit and opportunistic bandits [23] instead of the stochastic bandit to condition trade-off control on other system and node properties.

References

[1] How microsoft drives exabyte analytics on the world's largest yarn cluster. *https://azure.microsoft.com/en-us/blog/how-microsoft-drives-exabyte-analytics-on-the-world-s-largest-yarn-cluster/*.

[2] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

[3] Omar Besbes, Yonatan Gur, and Assaf Zeevi. Stochastic multi-armed-bandit problem with non-stationary rewards. *Advances in neural information processing systems*, 27, 2014.

[4] Robert Birke, Ioana Giurgiu, Lydia Y Chen, Dorothea Wiesmann, and Ton Engbersen. Failure analysis of virtual and physical machines: patterns, causes and characteristics. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 1–12. IEEE, 2014.

[5] Amit Biswas, Ashish Kumar Maurya, Anil Kumar Tripathi, and Samir Aknine. Frlle: a failure rate and load-based leader election algorithm for a bidirectional ring in distributed systems. *The Journal of Supercomputing*, 77(1):751–779, 2021.

[6] Francesco Bullo, Jorge Cortés, and Sonia Martinez. *Distributed control of robotic networks*. Princeton University Press, 2009.

[7] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[8] Michael Eischer and Tobias Distler. Latency-aware leader selection for geo-replicated byzantine fault-tolerant systems. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 140–145. IEEE, 2018.

[9] Daniel Ford, François Labelle, Florentina I Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan. Availability in globally distributed storage systems. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, 2010.

[10] Indranil Gupta, Robbert van Renesse, and Kenneth P Birman. A probabilistically correct leader election protocol for large groups. In *International symposium on distributed computing*, pages 89–103. Springer, 2000.

[11] Eshcar Hillel, Zohar S Karnin, Tomer Koren, Ronny Lempel, and Oren Somekh. Distributed exploration in multi-armed bandits. *Advances in Neural Information Processing Systems*, 26, 2013.

[12] Shay Kutten, William K Moses Jr, Gopal Pandurangan, and David Peleg. Singularly optimal randomized leader election. *arXiv preprint arXiv:2008.02782*, 2020.

[13] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, 2001.

[14] Leslie Lamport. Lower bounds for asynchronous consensus. *Distributed Computing*, 19(2):104–125, 2006.

[15] Peter Landgren, Vaibhav Srivastava, and Naomi Ehrich Leonard. Distributed cooperative decision making in multi-agent multi-armed bandits. *Automatica*, 125:109445, 2021.

[16] Barbara Liskov and James Cowling. Viewstamped replication revisited. 2012.

[17] Shengyun Liu and Marko Vukolić. Leader set selection for low-latency geo-replicated state machine. *IEEE Transactions on Parallel and Distributed Systems*, 28(7):1933–1946, 2016.

[18] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm (extended version), 2013.

[19] SungHoon Park, SuChang Yoo, and BoKyoung Kim. An election protocol based on group membership detection algorithm in mobile ad hoc distributed systems. *The Journal of Supercomputing*, 74(5):2239–2253, 2018.

[20] Murali Krishna Ramanathan, Ronaldo A Ferreira, Suresh Jagannathan, Ananth Grama, and Wojciech Szpankowski. Randomized leader election. *Distributed Computing*, 19(5):403–418, 2007.

[21] Ermin Sakic, Petra Vizarreta, and Wolfgang Kellerer. Seer: Performance-aware leader election in single-leader consensus. *arXiv preprint arXiv:2104.01355*, 2021.

[22] Pranav Sakulkar and Bhaskar Krishnamachari. Contextual combinatorial bandits in wireless distributed computing. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pages 543–547. IEEE, 2017.

[23] Huasen Wu, Xueying Guo, and Xin Liu. Adaptive exploration-exploitation tradeoff for opportunistic bandits. In *International Conference on Machine Learning*, pages 5306–5314. PMLR, 2018.

[24] Fei Zhang, Guangming Liu, Xiaoming Fu, and Ramin Yahyapour. A survey on virtual machine migration: Challenges, techniques, and open issues. *IEEE Communications Surveys & Tutorials*, 20(2):1206–1243, 2018.

[25] Gengrui Zhang and Hans-Arno Jacobsen. Prosecutor: an efficient bft consensus algorithm with behavior-aware penalization against byzantine attacks. In *Proceedings of the 22nd International Middleware Conference*, pages 52–63, 2021.

[26] Yiwen Zhu, Subru Krishnan, Konstantinos Karanasos, Isha Tarte, Conor Power, Abhishek Modi, Manoj Kumar, Deli Zhang, Kartheek Muthyala, Nick Jurgens, et al. Kea: Tuning an exabyte-scale data infrastructure. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2667–2680, 2021.