On the Analysis and Evaluation of Proximity Based Load Balancing Policies

NITISH K. PANIGRAHY, University of Massachusetts Amherst
THIRUPATHAIAH VASANTAM, Durham University
PRITHWISH BASU, Raytheon BBN Technologies
DON TOWSLEY, University of Massachusetts Amherst
ANANTHRAM SWAMI, Army Research Laboratory
KIN K. LEUNG, Imperial College London

Distributed load balancing is the act of allocating jobs among a set of servers as evenly as possible. The static interpretation of distributed load balancing leads to formulating the load balancing problem as a classical balls and bins problem with jobs (balls) never leaving the system and accumulating at the servers (bins). While most of the previous work in the static setting focus on studying the maximum number of jobs allocated to a server or *maximum load*, little importance has been given to the *implementation cost*, or the cost of moving a job/data to/from its allocated server, for such policies.

This paper designs and evaluates server proximity aware static load balancing policies with a goal to reduce the *implementation cost*. We consider a class of proximity aware Power of Two (POT) choice based assignment policies for allocating jobs to servers, where both jobs and servers are located on a two-dimensional Euclidean plane. In this framework, we investigate the tradeoff between the implementation cost, and load balancing performance of different allocation policies. To this end, we first design and evaluate a *Spatial Power of two* (sPOT) policy in which each job is allocated to the least loaded server among its two geographically nearest servers. We provide expressions for the lower bound on the asymptotic expected maximum load on the servers and prove that sPOT does not achieve classical POT load balancing benefits. However, experimental results suggest the efficacy of sPOT with respect to expected implementation cost. We also propose two non-uniform server sampling based POT policies that achieve the best of both implementation cost and load balancing performance.

We then extend our analysis to the case where servers are interconnected as an n-vertex graph $G(S, E)$. We assume each job arrives at one of the servers, u, chosen uniformly at random from the vertex set S. We then assign each job to the server with minimum load among servers u and v where v is chosen according to one of the following two policies: (i) Unif-POT(k): Sample a server v uniformly at random from k-hop neighborhood of u (ii) InvSq-POT(k): Sample a server v from k-hop neighborhood of u with probability proportional to the inverse square of the distance between u and v. An extensive simulation over a wide range of topologies validate the efficacy of both the policies. Our simulation results show that both policies consistently produce a load distribution which is much similar to that of a classical POT. Depending on topology, we observe the total variation distance to be of the order of $0.002 - 0.08$ for both the policies while achieving a $8\% - 99\%$ decrease in implementation cost as compared to the classical POT.

1 INTRODUCTION

The past few years have witnessed an increased interest in the use of large-scale parallel and distributed systems for database and commercial applications. An important design goal in such a system is to distribute service requests or jobs among servers or distributed resources as evenly as possible. While the optimal server selection problem can be solved centrally, due to scalability

Authors' addresses: Nitish K. Panigrahy, University of Massachusetts Amherst, Amherst, MA 01003, USA, nitish@cs.umass.edu; Thirupathaiah Vasantam, Durham University, Durham, DH13LE, UK, thirupathaiah.vasantam@durham.ac.uk; Prithwish Basu, Raytheon BBN Technologies, Cambridge, MA 02138, USA, prithwish.basu@raytheon.com; Don Towsley, University of Massachusetts Amherst, Amherst, MA 01003, USA, towsley@cs.umass.edu; Ananthram Swami, Army Research Laboratory, Adelphi, MD 20783, USA, ananthram.swami.civ@mail.mil; Kin K. Leung, Imperial College London, London SW72AZ, UK, kin.leung@imperial.ac.uk.

concerns, it is often preferred to adopt distributed randomized load balancing strategies to distribute these jobs among servers. This interpretation leads to formulating a randomized *load balancing* problem for the distributed systems with the goal to make the overall user-to-server assignment as fair as possible. Many previous works [1, 25] have used randomization as an effective tool to develop simple and efficient load balancing algorithms in non-geographic settings. A randomized load balancing algorithm can be described as a classical balls and bins problem as follows.

In the classical balls-and-bins model of randomized load balancing, m balls are placed sequentially into n bins. Each ball samples d bins uniformly at random and is allocated to the bin with the least number of balls, ties broken arbitrarily. It is well known that when $d = 1$ and $m = n$, this assignment policy results in a maximum load of $O(\log n/\log\log n)$ with high probability [3]. However, if $d = 2$, then the maximum load is $O(\log\log n)$ w.h.p. [3]. Thus, there is an exponential improvement in performance from $d = 1$ to $d = 2$. This policy with $d = 2$ is widely known as *Power of Two* (POT) choices and the improvement in maximum load behavior is known as *POT benefits* [25]. Many subsequent works have studied assignment policies that generalize POT policy to account for correlated and non-uniform sampling strategies [6], [10], [35].

1.1 Spatial Load Balancing

While classical balls and bins based randomized load balancing can directly be used for user/job to server assignment in a geographic setting, it is oblivious to the spatial distribution of servers and users. For example, in applications like Internet of Things [2], a large number of computational and storage resources are deployed in a physical space. These resources/servers are heavily accessed by various end users/applications that are also distributed across the physical space. The spatial distribution of servers and users is vital in determining the overall performance of the service network. We define the cost of moving jobs/results to/from their allocated servers as the *implementation cost* associated with a given policy. The implementation cost generally increases with the Euclidean distance between the user and its allocated server, also known as *allocation distance*. For example in wireless networks, signal attenuation is strongly coupled to allocation distance, therefore developing allocation policies that minimize allocation distance can help reduce energy consumption. Thus the following natural question arises.

> *How should we design proximity aware load balancing policies that also reduce overall policy implementation cost?*

In this work, we aim to answer this question. To this end we propose a spatially motivated POT policy: *spatial POT* (sPOT) in which each user is allocated to the least loaded server among its two geographically nearest servers. We assume both users and servers are placed in a two-dimensional Euclidean plane. When both servers and users are placed uniformly at random in the Euclidean plane, we map sPOT to a classical balls and bins allocation policy with bins corresponding to the Voronoi regions associated with the second order Voronoi diagram of the set of servers. We show that sPOT performs better than POT in terms of average allocation distance. However, a lower bound analysis on the asymptotic expected maximum load for sPOT suggests that POT load balancing benefits are not achieved by sPOT. Inspired by the analysis of sPOT, we further propose two assignment policies and empirically show that these policies achieve the best of both allocation distance and maximum load behavior.

1.2 Load Balancing On Graphs

While designing load balancing policies on a plane is important, load balancing algorithms for other settings such as arbitrary graphs have been studied and have applications in many fields including bike-sharing systems, World Wide Web, peer-to-peer networks, vehicular wireless ad-hoc

networks. Also, load balancing can prolong the lifetimes of battery-powered wireless sensor and actuator networks where energy is a limited resource [21].

Load balancing algorithms for certain fixed-degree deterministic graphs, in particular ring topologies, have been studied in [17], [34] and have applications in bike-sharing systems. While many complex networks like the World Wide Web or peer-to-peer networks can be modeled as scale free or random regular graphs [12], little is known about load balancing policies on such random networks. Moreover, other complex systems such as transportation and mobility networks are often best represented as spatial graphs where nodes and edges are embedded in Euclidean space. For example, the communication network resulting from radio transmitters and wireless devices can be described by a random geometric graph [27]. Such networks have natural notions of distance and the cost of assigning a job scales with distance. Hence it is important to take into account this geographical aspect when designing load balancing policies.

In the graph model, servers are represented as vertices of an arbitrary graph $G(S, E)$. When a job arrives at a server u (origin server), it is assigned to the server with the least load among server u and $d-1$ servers sampled uniformly at random from its neighborhood in G [22]. In order to make the graph model more tractable for theoretical analysis, many simplified assumptions have been made over the graph structure. For example, Kenthapadi et al. [22] studied the scaling of maximum load for the case $d = 2$ by allowing G to be regular or almost regular with degree n^{ϵ}. However, in practice, real world networks are highly irregular. Moreover, state-of-the-art lacks a comprehensive study on characterizing the implementation cost associated with load balancing policies on real world networks.

More often previous work only considers developing theoretical framework to characterize the scaling of the maximum load behavior. In some applications, studying distribution of load may be more important than the maximum load since it yields a better resolution into the load characteristics of the network. Also in some cases, the peak of the load distribution may be significantly lower than the maximum load for a given allocation policy. Similarly, while designing proximity aware load balancing policies, one may try to balance between two performance metrics: load and implementation cost. Therefore, while the maximum load can end up being higher in such policies than classical POT, if the distribution is nearly the same as POT one could consider such a policy to be better than POT in the sense of implementation cost, and almost as good as POT in the sense of load balancing. These discussions bring us down to the following research questions.

(1) How should one evaluate the performance of proximity aware load balancing policies for non-regular graph models, such as random, scale free or spatial graph structures?

(2) What is a good performance metric to characterize the implementation cost associated with real world complex networks?

(3) What is the effect on load and implementation cost if $d-1$ servers are sampled from a k-hop neighborhood instead of a one hop neighborhood with $k \geq 2$?

(4) How close is the performance of a proximity based policy to POT with respect to load distribution instead of maximum load metric?

One of the primary motivations behind this work is to address these questions. The key challenge in developing theoretical frameworks to answer these questions is that the notion of neighborhood for each job heavily depends on the choice of graph topology. Thus even asymptotic results for load balancing policies under generic graph model are scarce and techniques like witness tree methods [22] are not applicable. There is little hope of analyzing the graph model in this generality

and generating analytical insights seems difficult to achieve. For this reason we investigate this model through detailed and extensive computer simulations across a variety of graph topologies in Section 4.

We first propose a server proximity aware policy, *Unif-POT(k)*, as follows. For each job, a server v is sampled uniformly at random from the k-hop neighborhood of the origin server u. The job is then allocated to the server with the smallest load among u and v. Since a POT policy is stochastically optimal with respect to maximum load among all online policies that sample two servers uniformly at random (Corollary 8, [3]), we thus compare Unif-POT(k) policy to that of POT policy. We also propose another proximity based load balancing policy: *InvSq-POT(k)* as follows. For each job, a server v is sampled from the k-hop neighborhood of origin server u with probability proportional to the inverse square of the shortest path distance measured in number of hops between u and v. The job is then allocated to the server with the smallest load among u and v. Through extensive simulations we verify that such a simple modification in the sampling technique, produces load distribution behavior very similar to that of POT policy while drastically reducing the average implementation cost across a variety of network topologies.

1.3 Contributions

One of the main contributions of this paper is to study and develop load balancing policies that account for policy implementation cost when servers are placed on a plane and on vertices of a graph. Our contributions are summarized below.

- **Proximity based load balancing on a plane**

(1) We analyze sPOT yielding lower bound expressions for asymptotic expected maximum load.
(2) We prove that no POT benefit is observed when servers are placed either on a two dimensional square grid or uniformly at random on a plane.
(3) We develop two non-uniform server sampling based POT policies on a plane to improve load and allocation distance behavior.

- **Proximity based load balancing on a graph**

(1) Our simulations demonstrate a total variation distance as low as $0.002 - 0.005$ between load distributions of classical POT and proposed proximity based policies across a wide range of network topologies.
(2) We achieve a significant reduction in implementation cost on the order of $20\% - 99\%$ for proposed proximity based policies as compared to classical POT.
(3) We observe that InvSq-POT(k) with $k = O(\log n)$ achieves the best of both the worlds, i.e. it is better than Unif-POT(k) in terms of implementation cost but only slightly worse than classical POT in terms of load distributions. To our surprise, even a very local sampling, i.e., InvSq-POT(k) with $k = O(1)$ achieves a load balancing performance almost similar to that of classical POT for certain random network topologies.

The rest of this paper is organized as follows. The next section contains some technical preliminaries. In Section 3 we formally analyze the load behavior of sPOT for different server placement settings on an Euclidean plane and propose two non-uniform server sampling based POT policies that achieve both better load and allocation distance behavior. In Section 4, we evaluate the performance of our proposed proximity based load balancing policies on arbitrary graphs. A summary of related literature is given in Section 5. Finally, the conclusion of this work and potential future work are given in Section 6.

2 TECHNICAL PRELIMINARIES

In this section, we introduce the system model used in the rest of the paper. We denote the users/jobs[1] in the system as the set J with $|J| = m$. Similarly, denote S as the set of servers with $|S| = n$. Let $\pi : J \rightarrow S$, denotes a load balancing policy for assigning users/jobs to servers. We generally assume $m = n$. However, in Section 4.6 , we also consider the case when $m > n$ and $m < n$.

2.1 Load balancing on a plane

We first consider a service network where users and servers are located on a two-dimensional Euclidean plane $\mathcal{D}$. We assume users are placed on a two-dimensional euclidean plane uniformly at random. In the service network, each user is assigned to a server from the server set S. We consider two cases for placing the servers on a two-dimensional euclidean plane, (i) Grid Placement: servers are placed on a square grid topology embedded in Euclidean space $\mathbb{R}^2$. (ii) Uniform placement: servers are placed uniformly at random on the euclidean plane.

We define the following geometric structures that are useful constructs for analyzing various load balancing policies on a plane.

Definition 1. *Voronoi Diagram: A Voronoi cell around a server $s \in S$ is the set of points in $\mathcal{D}$ that are closer to s than to any other server in $S \setminus \{s\}$ [5]. The Voronoi diagram V_S of S is the set of Voronoi cells of servers in S.*

Definition 2. *Delaunay Graph: The Delaunay graph, $G_S(S, E)$, is associated with the set of servers S. Here $(u, v) \in E$ iff the Voronoi cells of $u, v \in S$ are adjacent.*

Definition 3. *Higher order Voronoi diagram: A p^{th} order Voronoi diagram, $H_S^{(p)}$, is defined as partition of $\mathcal{D}$ into regions such that points in each region have the same p closest servers in S.*

In this paper, our goal is to analyze the performance of several load balancing policies on a plane including the two classic policies.

- **Power of One (*POO*):** This policy assigns each user to one of the servers chosen uniformly at random from S.
- **Power of Two (*POT*):** In this policy, sequentially each user samples two servers uniformly at random from S and is allocated to the least loaded server.

In addition we propose new policies to reduce both maximum load and expected allocation distance. We define them as follows.

- **InvSq-POT(k):** In this policy each user j samples two servers from a candidate set consisting of its k geographically nearest servers (without replacement), each with probability proportional to $1/d(j, v)^2$. Here, $d(j, v)$ denotes the euclidean distance between user j and server v. The user is then assigned to the least loaded server.
- **Unif-POT(k):** Each user uniformly at random samples two servers from a candidate set consisting of its k geographically nearest servers and is assigned to the least loaded server.

For our analysis we consider special cases of Unif-POT(k) policies with $k = 2$ and $k = 1$. We call them Spatial Power of Two (*sPOT*) and Spatial Power of One (*sPOO*) policies respectively for brevity. To be precise these policies are defined as

- **Spatial Power of Two (*sPOT*):** Each user is sequentially allocated to the least loaded server among its two geographically nearest servers.
- **Spatial Power of One (*sPOO*):** This policy assigns each user to its geographically nearest server.

[1]We use the terms "users" and "jobs" interchangeably.

2.2 Load balancing on a graph

For the graph model, we assume servers in the network are nodes of a connected graph $G(S, E)$ with $|S| = n$ and E a set of edges connecting the servers. We explore various random, deterministic and spatial graph structures for graph based load balancing systems. We assume that jobs arrive at one of the servers uniformly at random. Denote u as the arrival (origin) server for a job.

Next we define several network attributes that will be useful in analyzing the simulation results obtained for different load balancing policies later. We denote $d(u, v), u, v \in S$ as the shortest path distance measured in number of hops between nodes u and v in the network.

Definition 4. *k-hop Neighborhood: The k-hop neighborhood of a node $u \in S$ is defined as*

$$\mathcal{N}_k(u) = \{w | 1 \leq d(u, w) \leq k\}.$$

Definition 5. *Graph Density: The graph density of an undirected graph $G(S, E)$ is*

$$\rho_G = \frac{|E|}{\binom{n}{2}} = \frac{2|E|}{n(n-1)}.$$

Definition 6. *Average Path Length: The average path length of an undirected graph $G(S, E)$ is*

$$l_G = \frac{1}{n(n-1)} \sum_{u \neq v} d(u, v).$$

We now introduce the three load balancing policies that we study. Suppose a job arrives at origin server $u \in V$. Denote $P_u = [p_{uv}, v \in V]$ as the server sampling distribution for the job where p_{uv} is the probability u queries server v for its load information with $p_{uu} = 0$. The first policy is the well known POT policy and the next two are newly proposed proximity based load balancing policies on a graph G.

- **Power of Two (*POT*):** If a job arrives at server u, then

$$p_{uv} = \begin{cases} \frac{1}{n-1}, & \text{if } v \neq u, \\ 0, & \text{otherwise .} \end{cases} \tag{1}$$

  That is, server v is sampled uniformly at random from the remaining $n - 1$ servers. The job is then allocated to the server with the smallest load among u and v.

- **InvSq-POT(k):** According to this policy, if a job arrives at server u, then

$$p_{uv} = \begin{cases} \frac{\left(\frac{1}{d(u,v)^2}\right)}{\sum_{w \in \mathcal{N}_k(u)} \left(\frac{1}{d(u,w)^2}\right)}, & \text{if } v \in \mathcal{N}_k(u), \\ 0, & \text{otherwise .} \end{cases} \tag{2}$$

  That is, a server $v \in \mathcal{N}_k(u)$ is sampled with probability proportional to the inverse square of the distance to u. The job is then allocated to the server with the smallest load among u and v.

- **Unif-POT(k):** According to this policy, if a job arrives at server u, then

$$p_{uv} = \begin{cases} \frac{1}{|\mathcal{N}_k(u)|}, & \text{if } v \in \mathcal{N}_k(u), \\ 0, & \text{otherwise .} \end{cases} \tag{3}$$

That is, a server v is sampled uniformly at random from the k-hop neighborhood of u[2]. The job is then allocated to the server with the smallest load among u and v.

Remark 1. *Observe that Unif-POT(k) and InvSq-POT(k) are identical for $k = 1$. Similarly, POT and Unif-POT(k) are identical for $k = n$.*

2.3 Performance Metrics

To evaluate and characterize the performance of various load balancing policies, we define the performance metrics for both plane and graph based systems as follows.

Denote $x^\pi(t) = [x_i^\pi(t), i \in \{1, \cdots, m\}]$ as the state of the system immediately after the t^{th} job is assigned under policy π. Here $x_i^\pi(t)$ denotes the fraction of servers with exactly i jobs immediately after t^{th} job is assigned. Denote $x^\pi(m)$ as the load distribution under policy π after all of m jobs are assigned.

Definition 7. *Maximum Load: The maximum load for policy π is defined as*

$$ML^\pi = i \text{ with } x_i^\pi(m) \neq 0 \text{ and } x_j^\pi(m) = 0 \text{ for } j = i+1, \cdots, m.$$

Definition 8. *Total Variation Distance: The total variation distance between two load distributions $x^{\pi_1}(m)$ and $x^{\pi_2}(m)$ is*

$$TV^{\pi_1\pi_2} = \frac{1}{2}\sum_{i=1}^{m} |x_i^{\pi_1}(m) - x_i^{\pi_2}(m)|.$$

$TV^{\pi_1\pi_2}$ takes values in $[0, 1]$. The closeness of two load distributions under two different policies can be measured by the total variation distance, i.e. smaller the total variation distance the closer the two distributions are to each other.

Definition 9. *Average Allocation Distance[3]: The average allocation distance for policy π is the average distance (or number of hops) between a random user (or its origin server) and its allocated server under π, i.e.*

$$RD^\pi = \frac{1}{m}\sum_{j \in J} d(j, \pi(j)).$$

Since POT is oblivious to inter server distances, RD^{POT} is generally large as compared to other proximity based load balancing policies.

3 PROXIMITY AWARE POT POLICIES ON A PLANE

We now analyze the load behavior of sPOT policy for various server placements on a plane. We assume users are placed uniformly at random on $\mathcal{D}$.

3.1 sPOT with Grid based server placement

Consider the case where servers are placed on a two dimensional square grid: $\sqrt{n} \times \sqrt{n}$ on $\mathcal{D}$ with wrap-around. Let $B(\{s_1, s_2\}, r)$ be the event that the two nearest servers of r are in $\{s_1, s_2\}$. We prove the following Lemma.

[2]While one can sample v from k nearest servers as done for policies on a plane, choosing from the k-hop neighborhood simplifies the graph based model and does not require tie breaking mechanism to determine the sampling space.

[3]We assume the load information is fairly compact, and the overhead of propagating this information is far less than migrating the job/data from one node/user to the allocated server. Even though we may request the load values from many servers, distances to servers to whom we ended up not assigning the request matter less.

Lemma 1. *Let $G_S(X, E)$ denote the Delaunay graph associated with S when servers are placed on a two-dimensional square grid. Then*

$$\Pr[B(\{s_i, s_j\}, r)] = \begin{cases} \frac{1}{|E|}, & (s_i, s_j) \in E; \\ 0, & \textit{otherwise.} \end{cases} \tag{4}$$

Proof. Let $A(s, r, l)$ denote the event that a random user $r \in J$ is l^{th} closest to $s \in S$ among all servers in S. Denote $NN(r)$ as the geographically nearest server of r. Thus we have

$$\begin{aligned} \Pr[B(\{s_i, s_j\}, r)] = &\Pr[A(s_i, r, 1)] \Pr[A(s_j, r, 2) | NN(r) = s_i] \\ &+ \Pr[A(s_j, r, 1)] \Pr[A(s_i, r, 2) | NN(r) = s_j]. \end{aligned} \tag{5}$$

It is not difficult to show that all Voronoi cells in V_S have equal areas. As $\Pr[A(s, r, 1)]$ is proportional to the area of the Voronoi cell surrounding s, we have

$$\Pr[A(s, r, 1)] = 1/|S| \ \forall \ s \in S. \tag{6}$$

Without loss of generality (*W.l.o.g.*) consider a user r placed uniformly at random on $\mathcal{D}$ as shown in Figure 1. Denote $\triangle ABC$ as the triangle associated with vertices A, B and C. Let $NN(r) = s_3$. We now evaluate $\Pr[A(s_1, r, 2) | NN(r) = s_3]$. Clearly, $\Pr[A(s_1, r, 2) | NN(r) = s_3] \propto \text{Area}(\triangle WXs_3)$. We also have

$$\begin{aligned} \text{Area}(\triangle WXs_3) &= \text{Area}(\triangle WZs_3) = \text{Area}(\triangle YXs_3) \\ &= \text{Area}(\triangle ZYs_3), \end{aligned}$$

Therefore $\Pr[A(s_i, r, 2) | NN(r) = s_3]$, for $i \in \{1, 2, 4, 5\}$ are all equal. Let $NG(s)$ be the set of neighboring servers of a server $s \in S$ on the square grid. Thus, we have

$$\Pr[A(s_j, r, 2) | NN(r) = s_i] = \begin{cases} \frac{1}{4}, & s_j \in NG(s_i); \\ 0, & \text{otherwise}, \end{cases} \tag{7}$$

Note that when $s_j \in NG(s_i)$, the Voronoi cells corresponding to s_i and s_j share an edge. In this case, by definition $(s_i, s_j) \in E$. Combining (6) and (7) and substituting in (5) yields

$$\Pr[B(\{s_i, s_j\}, r)] = \begin{cases} \frac{1}{2|S|}, & (s_i, s_j) \in E; \\ 0, & \text{otherwise}, \end{cases} \tag{8}$$

Also, when servers are placed on a square grid, $G_S(X, E)$ is 4- regular. Thus the total number of edges is $|E| = 2|X| = 2|S|$. Substituting $|S| = |E|/2$ in Equation (8) yields (4) and completes the proof. □

We consider the following lemma presented in [20].

Lemma 2. *Given a Δ-regular graph with n nodes representing n bins, if n balls are thrown into the bins by choosing a random edge and placing into the smaller of the two bins connected by the edge, then the maximum load is at least $\Omega(\log\log n + \frac{\log n}{\log(\Delta \log n)})$ with high probability of $1 - 1/n^{\Omega(1)}$.*

We now prove the following theorem.

Theorem 1. *Suppose servers are placed on a two dimensional square grid : $\sqrt{n} \times \sqrt{n}$ on $\mathcal{D}$ with wrap-around. Let users be placed independently and uniformly at random on $\mathcal{D}$. Under sPOT, the maximum load over all servers is at least $\Omega(\frac{\log n}{\log\log n})$ with high probability of $1 - 1/n^{\Omega(1)}$.*

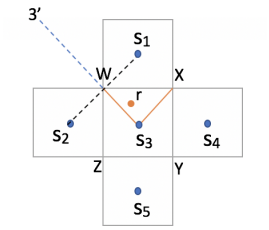


Fig. 1. Second order Voronoi regions for servers placed on a grid.

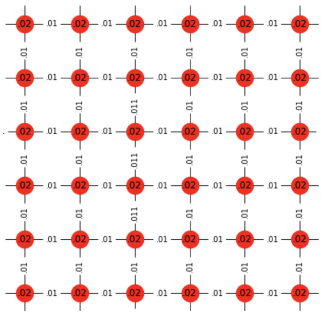


Fig. 2. Delaunay Graph associated with grid based server placement.

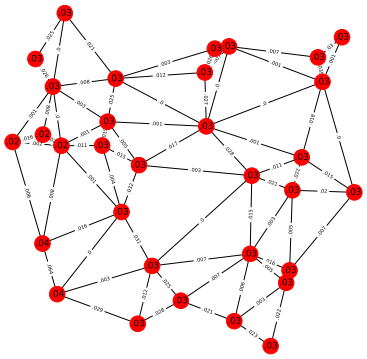


Fig. 3. Delaunay Graph associated with uniform server placement.

Proof. Suppose we map the set of servers to the bins and the users to the balls. The delaunay graph G_S is 4-regular. Let $e = (s_i, s_j)$ be an edge in G_S. From Lemma 1, it is clear that each user (ball) selects an edge e with probability $1/|E|$ (i.e. uniformly at random) and gets allocated to the smaller of the two servers (bins) connected by e under sPOT policy. Thus a direct application of Lemma 2 with $\Delta = 4$ proves the theorem. □

We verify the results in Lemma 1 through simulation for a 2D square grid under sPOT as shown in Figure 2. We assign $n = 64$ and empirically compute $\Pr[B(\{s_i, s_j\}, r)]$ and denote it as edge probability on edge e on the Delaunay graph. We also verify the $\Pr[A(s, r, 1)]$ in expression (6) and denote it as vertex probability on the Delaunay graph. It is clear from Figure 2 that the edge probabilities are almost all equal and so are the vertex probabilities.

Remark 2. *Note that, the maximum load associated with the POO policy is* $\frac{\log n}{\log \log n} + O(1)$. *Thus, Theorem 1 concludes that we do not get POT benefits under sPOT and it performs no better than POO when servers are placed on a two dimensional square grid.*

Remark 3. *Note that Theorem 1 applies to other grid graphs such as a triangular grid, i.e. we do not get POT benefits when servers are placed on a two dimensional triangular grid. The delaunay graph corresponding to a triangular grid based server placement is* 6*- regular.*

3.2 sPOT with Uniform server placement

We now consider the case when both users and servers are placed uniformly at random on $\mathcal{D}$. We can no longer invoke Lemma 2. This is due to the fact that the Delaunay graph associated with the servers is no longer regular. Also, the edge sampling probabilities $\Pr[B(s_i, s_j, r)]$ are no longer equal. This is evident from our simulation results on the corresponding Delaunay graph as shown in Figure 3. We have $n = 32$ servers placed randomly in a 2D square and empirically compute $\Pr[B(\{s_i, s_j\}, r)]$ and denote it as edge probability on edge e on the Delaunay graph. Note that the edge probabilities, i.e. $\Pr[B(\{s_i, s_j\}, r)]$, differ from each other. Also the Delaunay graph is not regular. Thus we resort to using a second order Voronoi diagram to analyze the maximum asymptotic load behavior.

3.2.1 Majorization Basics. We present a few definitions and basic results associated with majorization theory that we apply to analyze sPOT.

Definition 10. *The vector x is said to majorize the vector y (denoted $x \succ y$) if*

$$\sum_{i=1}^{k} x_{[i]} \geq \sum_{i=1}^{k} y_{[i]},\ k = 1, \cdots, n-1,$$
$$and \sum_{i=1}^{n} x_{[i]} = \sum_{i=1}^{n} y_{[i]} \tag{9}$$

where $x_{[i]}(or\ y_{[i]})$ is the i^{th} largest element of $x(or\ y)$.

Definition 11. *A function $f : R^n \rightarrow R$ is called Schur-convex if*

$$x \succ y \implies f(x) \geq f(y) \tag{10}$$

Consider the following proposition.

Proposition 1. *(Chapter 11, [24]) Let X be a random variable having the multinomial distribution*

$$\Pr[X = x] = \binom{n}{x_1, \cdots, x_n} \prod_{i=1}^{n} p_i^{x_i} \tag{11}$$

where $x = (x_1, ..., x_n) \in \{z : z_i$ are nonnegative integers, $\sum z_i = n\}$. If δ is a Schur-convex function of X, then $\psi(p) = E_p\delta(X)$ is a Schur-convex function of p.

3.2.2 Loss of POT benefits under sPOT. Consider the second order Voronoi diagram: $H_S^{(2)}$ associated with the set of servers S. We have the following Lemma [Chapter 3.2, [26]].

Lemma 3. *The number of Voronoi cells in $H_S^{(2)}$ under uniform server placement is upper bounded by $O(3n)$.*

We also have the following Lemma.

Lemma 4. *Consider the following modified version of balls and bin problem. Suppose there are n balls and n bins. Each ball is thrown into one of the bins according to a probability distribution $p = (p_1, \cdots, p_n)$ with p_i being the probability of each ball falling into bin i, in an independent manner. Denote Z to be the random variable associated with the maximum number of balls in any bin. Then we have*

$$E_p[Z] \geq k_0 \frac{\log n}{\log \log n} \text{ as } n \rightarrow \infty. \tag{12}$$

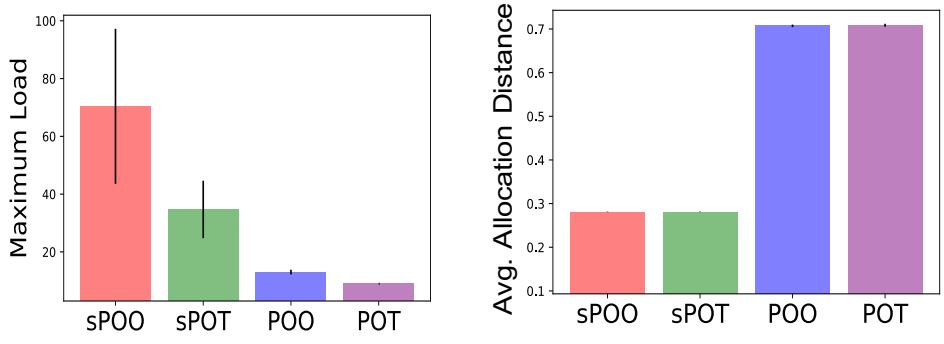
where k_0 is a scalar constant.

Proof. Denote X_i as the random variable associated with the load for bin i. Clearly $X = [X_1, X_2, \cdots, X_n]$ follows multinomial distribution

$$\Pr[X = x] = \binom{n}{x_1, \cdots, x_n} \prod_{i=1}^{n} p_i^{x_i} \tag{13}$$

We have $Z = \max(X_1, \cdots, X_n) = \delta(X)$. Clearly, $\delta(x) = \max(x)$ is a schur convex function since $\max(x) = x_{[1]}$ and if $x \succ y$ then $x_{[1]} \geq y_{[1]}$. Also, we have (Chapter 1, [24]): $(p_1, p_2, \cdots, p_n) \succ (1/n, 1/n, \cdots, 1/n)$. whenever $p_i \geq 0$ with $\sum_{i=1}^{n} p_i = 1$. Thus applying Proposition 1 yields $E_p[Z] \geq E_{(1/n, \cdots, 1/n)}[Z] \geq k_0 \frac{\log n}{\log \log n}$. □

Theorem 2. *Suppose both users and servers are placed independently and uniformly at random on $\mathcal{D}$. Under sPOT, the expected maximum load over all servers is at least $\Omega(\frac{\log n}{\log \log n})$ with high probability of $1 - 1/n^{\Omega(1)}$, i.e., we do not get POT benefits.*



(a) Maximum load

(b) Expected allocation distance

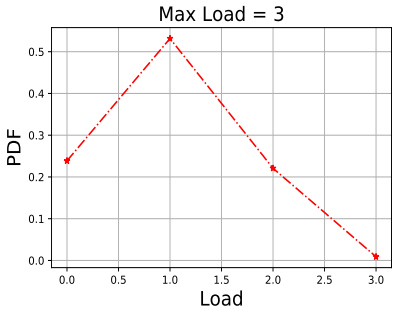
Fig. 4. Performance comparison of basic allocation policies wrt (a) maximum load and (b) expected allocation distance for $n = 10000$ servers.

Proof. Consider the second order Voronoi diagram: $H_S^{(2)}$ associated with the set of servers S. W.l.o.g. consider a cell $\{s_i, s_j\}$ in $H_S^{(2)}$. The probability that a user selects the server pair $\{s_i, s_j\}$ as its two nearest servers is proportional to the area of the cell $\{s_i, s_j\}$. However, the area distribution of cells in $H_S^{(2)}$ is non-uniform (say with probability distribution p). We can invoke classical balls and bins argument on $H_S^{(2)}$ as follows inspired by the discussion in [20]. We treat each cell in $H_S^{(2)}$ as a bin. Thus by Lemma 3, there are $O(3n)$ bins (or cells). Each ball (or user) choses a bin (or a cell) from a distribution p and let L_p denote the expected maximum asymptotic load across the bins. Let L_U denote the expected maximum asymptotic load across the bins when $O(n)$ balls are assigned to $O(3n)$ bins with each ball choosing a bin uniformly at random. From classical balls and bins theory, $L_U = O(\log n/\log\log 3n) = O(\log n/\log\log n)$. Clearly, by Lemma 4, we have $L_p \geq L_U = O(\log n/\log\log n)$. Since a cell consists of a server pair, one of the server pair corresponding to the maximum load would have load at least $(1/2)L_p$. Thus the maximum load across all servers would be at least $(1/2)L_p \geq O(\log n/\log\log n)$. □

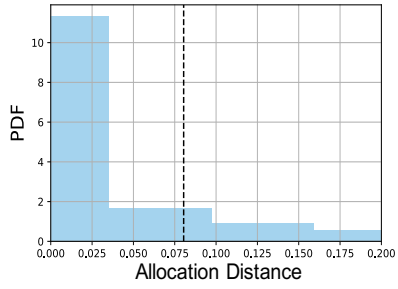
3.3 Tradeoff between Load and Allocation Distance

In this Section, we discuss the inherent tradeoff between maximum load and expected allocation distance metric among different allocation policies. We evaluate the performance of sPOT and compare it to that of other allocation policies. We consider $n = 10000$ servers and an equal number of users placed on a unit square uniformly at random. We ran 10 trials for each policy. We compare the performance of various allocation policies in Figure 4. The black lines on top of the bars in Figures 4 (a) and (b) represent the standard deviation of maximum load and expected allocation distance associated with different allocation policies.

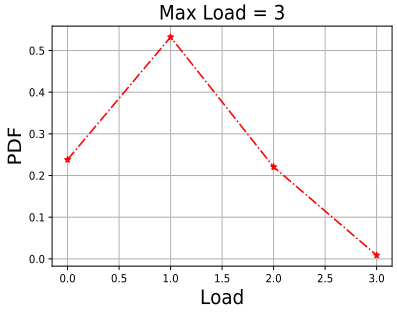
First, note that with respect to maximum load, the spatial based policies perform worse compared to their classical counterparts, POO and POT. Note that the introduction of spatial aspects into a policy increases its maximum load. For example, sPOT performs worse than both POO and POT as shown in Figure 4 (a). The scaling laws for the maximum load for sPOT and POO are $\Omega(\log n/\log\log n)$ and $O(\log n/\log\log n)$, respectively. Figure 4 (a) match this finding and validates our lower bound results obtained for sPOT in Theorem 2. Note that in POO, there is an equal probability of choosing any server. Since the areas of voronoi regions associated with servers are not equal, few servers are more likely to be picked-up by sPOO than others. Similarly, in sPOT,



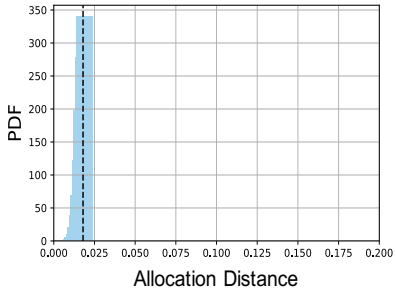
(a) InvSq-POT(n) Load distribution



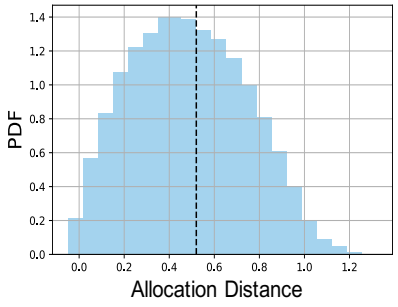
(b) InvSq-POT(n) alloc. dist. distribution



(c) POT Load distribution



(d) sPOT alloc. dist. distribution



(e) POT alloc. dist. distribution

Fig. 5. Performance comparison of allocation policies wrt InvSq-POT(n) for $n = 50000$ servers. (a) and (b) plots are for InvSq-POT(n) while (c),(e) and (d) for POT and sPOT respectively.

some servers are probably not going to be sampled at all since they are never among the nearest two servers of any point where the requests arrived at. This maybe true especially for a moderate number of arrivals e.g., 10000.

However, the expected allocation distance is smallest for sPOO and almost similar to that of sPOT. Also, both POT and POO have very large and similar expected allocation distances as shown in Figure 4 (b). Both results shown in Figure 4 (a) and (b) combined, illustrate the tradeoff between maximum load and expected allocation distance metric.

3.4 Improving Performance of Planar Policies

In previous sections, we showed a tradeoff between maximum load and expected allocation distance among different allocation policies. Note that for each user, once its arrival location is fixed, the choice of two servers by sPOT is deterministic while it is completely random for POT. This random sampling over the entire set of servers results in better load behavior for POT than for sPOT. However, since random sampling is oblivious to the distances of servers from the particular user, POT exhibits a large expected allocation distance. Thus if one can design a policy with random and distance dependent sampling of servers, such a policy should provide benefits of both POT and sPOT in terms of maximum load and expected allocation distance. Below we propose and evaluate two such policies that get benefits of both POT and sPOT. We empirically show that they achieve both POT like load benefits while having a allocation distance profile similar to that of sPOT. All simulations in this section assume servers and an equal number of users placed on a unit square uniformly at random.

3.4.1 InvSq-POT(k). Consider the allocation of a random user j in the service network. We propose InvSq-POT(k) to allocate j as follows. Under InvSq-POT(k), j samples two servers from a candidate set consisting of its k geographically nearest servers (without replacement), each with probability proportional to $1/d(j,v)^2$. Here, $d(j,v)$ denotes the euclidean distance between user j and server v. User j is then allocated to the server with the least load among the two sampled servers. This rule is similar to one used in small world routing [23]. Note that, since the probability of sampling a server is inversely proportional to its distance from the user, InvSq-POT(k) incurs a smaller expected allocation distance compared to POT. Surprisingly, InvSq-POT(k) achieves similar load behavior to that of POT. We compare the performance of InvSq-POT(k) to sPOT and POT as follows.

We perform 10 independent simulation runs for each of the policies: InvSq-POT(n), sPOT, POT and take the average of different performance metrics to generate plots in Figures 5 (a) - (e). We define the load associated with a server to be the number of users assigned to it. We measure the load distribution across all servers and the allocation distance distribution. Figure 5 (a) shows the load distribution and Figure 5 (b) shows the allocation distance distribution for InvSq-POT(n). We plot the load distribution for POT and allocation distance distribution for sPOT in Figure 5 (c) and (d) respectively.

First we focus on the server loads in Figure 5 (a) and (c). Interestingly, the load distributions are almost identical for InvSq-POT(n) and POT. Similarly, sPOT performs better than InvSq-POT(n) in terms of allocation distance distribution as shown in Figure 5 (b) and (d) since they significantly favor closer nodes. However, compared to POT (as shown in Figure 5 (e)), InvSq-POT(n) performs significantly better in terms of allocation distances. Thus InvSq-POT(n) achieves the best of both worlds, i.e., small maximum load and small allocation distances.

3.4.2 Unif-POT(k). We now propose a policy that improves the load behavior of sPOT. We define C_k to be the candidate set (of size k) consisting of the k nearest servers for a particular user. Under Unif-POT(k), the user selects two servers uniformly at random from C_k and assigns itself to the least loaded one. Note that, random sampling of two servers within the candidate set helps to balance load and reduce the overall maximum load. Clearly sPOT and POT are two extremes of the policy Unif-POT(k) with $k = 2$ and $k = n$ respectively. Below, we discuss the effect of k on maximum load and expected allocation distance and compare it to other policies. We perform 50 independent simulation runs for each of the policies: InvSq-POT(n), sPOT, POT, Unif-POT($\log n$), and take the average of different performance metrics to generate plots in Figures 6 (a) - (d).

We present total variation distance between load distributions of proximity based policies and POT as a function of number of servers in Figure 6 (a). We also plot the total variation distance

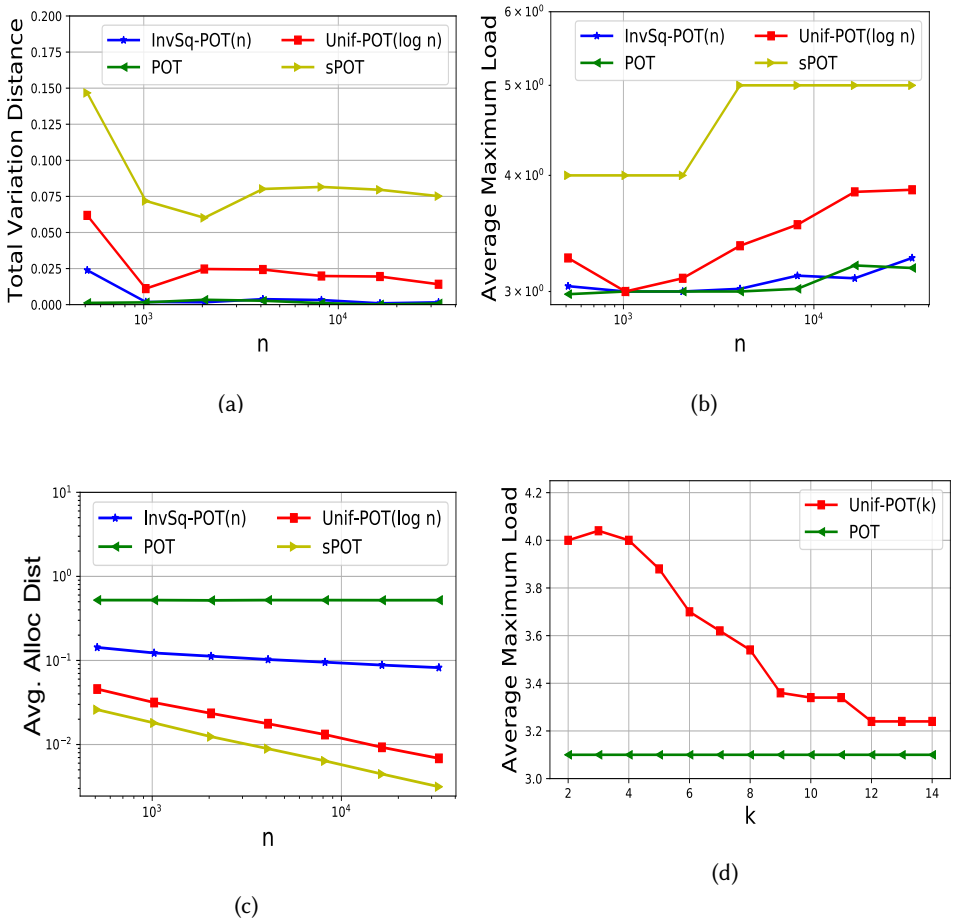


Fig. 6. Performance comparison of Unif-POT(k) and InvSq-POT(n) with respect to (a) total variation distance to POT (b) average maximum load and (c) average allocation distance when servers placed uniformly at random on a plane. (d) Performance of Unif-POT(k) for different values of k.

between load distributions of two independent runs of POT which quantifies the noise or variation in load distribution of POT due to randomness. Both Unif-POT($\log n$) and InvSq-POT(n) achieve total variation distances as low as 0.02 across a wide range of values of n. Also note that, sPOT achieves a load distribution farthest from POT while InvSq-POT(n) achieves the closest. Due to load-implementation cost trade-off, a very local policy sPOT, achieves larger variation distance.

Figure 6 (b) shows the growth in the average maximum load as n is varied. We observe that both InvSq-POT(n) and POT perform the best. Unif-POT(k) with $k = \log n$ performs quite well compared to sPOT.

Figure 6 (c) shows how the average allocation distance drops as n increases (since the node density increases). We observe that, not surprisingly, sPOT outperform the rest. However, Unif-POT(k) with $k = \log n$ performs quite well. Thus Unif-POT(k) with $k = O(\log n)$ achieves good performance for both load and allocation distance.

Last, we plot the average maximum load as a function of k for Unif-POT(k) policy in Figure 6 (d). We have considered a total of $n = 10000$ number of servers and an equal number of users placed uniformly at random on a plane. We find that the average maximum load is high for Unif-POT(k)

as compared to POT and no POT benefit is observed for fixed values of k. However, the average maximum load decreases with increase in k and is within 4% difference of that of POT when $k = \log n$. Based on this result, we present the following conjecture.

Conjecture 1. *If the candidate set in Unif-POT(k) does not grow with n, no POT benefit is expected.*

Remark 4. *Among the proximity aware POT policies on a plane, InvSq-POT(n) selects servers through distance based sampling. Thus even a minor change in positions of servers theoretically requires choosing a new set of sampling distributions. However, sampling in Unif-POT(*$\log n$*) depends on the local neighborhood of the user and thus involves less frequent updates for server sampling distributions.*

3.5 Effect of User Dynamics

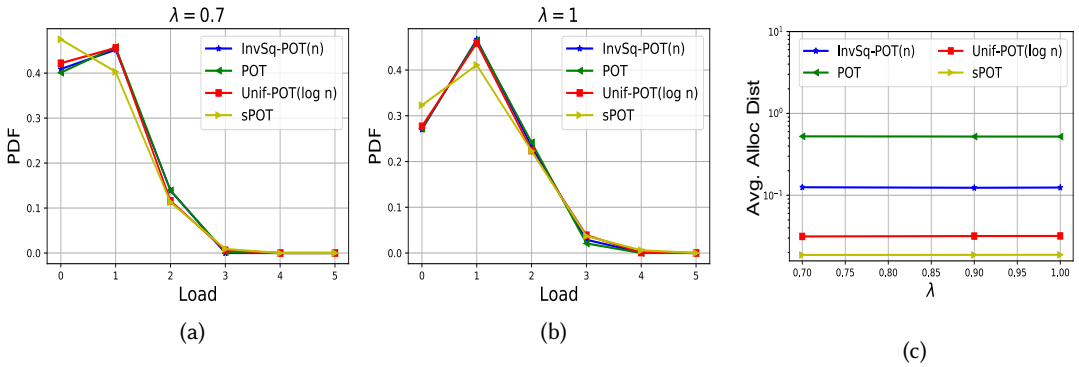


Fig. 7. Performance of dynamic load balancing policies for uniform server placements on a 2D plane with $n = 1000$ servers and 10^7 user arrivals.

So far we have considered a static load balancing system, i.e. the users never depart and the requests just accumulate at the servers. However, our proposed policies are still well defined and applicable for dynamic load balancing systems, i.e. in systems where incoming user requests are assigned to one of the servers and then they leave the system after getting served according to a service discipline.

In this section, we consider a dynamic load balancing system where users arrive on a two-dimensional Euclidean plane uniformly at random. We assume the inter-user arrival times are exponentially distributed with rate $n\lambda$. The servers are assumed to be placed on a two dimensional plane uniformly at random. Users are allocated to servers according to different proximity aware POT policies. Each server follows an M/M/∞ queuing model where every user request arrival on a server experiences immediate service and does not wait. Service times at servers are independent and identically distributed (IID) exponential random variables with mean μ. A processed user departs the system upon completion of its service.

For our simulations, we consider $n = 1000$ servers with 10^7 number of user arrivals. We assume $\mu = 1$. We plot the load (the number of users being served at any moment) distribution for different policies in Figures 7 (a) and (b). We observe that both InvSq-POT(n) and UnifPOT($\log n$) achieve load distributions which are very close to that of POT across different values of λ. Note that, the load distribution obtained by sPOT is quite different compared to POT and performs the worst. We also plot average allocation distance for various policies across different values of λ as shown in

Graph Type	n	m	Parameters
Line	$[1000, \cdots, 7000]$	$[1000, \cdots, 7000]$	$n : [1000, \cdots, 7000]$
Ring	$[1000, \cdots, 7000]$	$[1000, \cdots, 7000]$	$n : [1000, \cdots, 7000]$
Erdős Rényi (n, γ)	10000	10000	$\gamma : [\log n/n, \cdots, 2\log n/n]$
Random Regular (n, β)	10000	10000	$\beta : [5, 6, \cdots, 11]$
Barabasi Albert (n, α)	10000	10000	$\alpha : [1, 2, \cdots, 7]$
Random Geometric (n, r)	10000	10000	$r : [\sqrt{\log n/\pi n}, \cdots, \sqrt{\sqrt{n}/\pi n}]$
Spatial Line (n, L_{max})	$[1000, \cdots, 7000]$	$[1000, \cdots, 7000]$	$L_{max} : [1000, \cdots, 7000]$
Spatial Ring (n, R)	$[1000, \cdots, 7000]$	$[1000, \cdots, 7000]$	$R : 1$

Table 1. Simulation parameters for different network topologies.

Figure 7 (c). We observe similar results as that of the static scenario, i.e. sPOT achieves the lowest average allocation distance while POT has the highest. UnifPOT($\log n$) achieves the second best average allocation distance, closer to that achieved by sPOT.

Remark 5. *The results presented in this section are preliminary and not exhaustive. For example, when servers follow M/M/1 queueing model, each server is shared among users and competition among users for server resource may impact the performance of proposed load balancing policies. A comprehensive analysis and evaluation of proximity aware dynamic load balancing policies for different queueing models such as M/M/1 and other service disciplines remain one of our future works.*

4 PROXIMITY AWARE POT POLICIES ON GRAPHS

In Sections 3, we proposed new load balancing policies for the case when users/jobs and servers are distributed on a two dimensional Euclidean plane. However, load balancing algorithms for more general settings, such as on a general graph have applications in many fields including bike-sharing systems, World Wide Web, peer-to-peer networks, vehicular wireless ad-hoc networks. We can easily extend the definitions of Unif-POT(k) and InvSq-POT(k) to a generic graph setting as mentioned in Section 2.2.

4.1 Experimental Setup

4.1.1 Investigated Policies and Performance Metrics. In this section we present extensive simulation results to illustrate the effectiveness of both Unif-POT(k) and InvSq-POT(k) policies in graph based load balancing systems. Our study also provides insights into the choice of a load balancing policy under different load conditions and for different network topologies. We evaluate the proposed schemes (POT, Unif-POT($\log n$), InvSq-POT($\log n$) and InvSq-POT(n)) using total variation distance, average allocation distance and average maximum load as performance metrics.

We implemented the proposed policies in Python to study their performance in a simulated environment. To make the performance comparisons between the algorithms meaningful, a number of simulation runs were conducted for each algorithm with different parameter values (e.g., system size, average degree etc.) for different graph topologies. If not specified, we assume $n = 10000$ servers interconnected through a graph G. Also, $m = 10000$ jobs each arriving sequentially to one of the servers chosen uniformly at random and is allocated to a server according to different proximity based POT policies. We report the average of 10 simulation runs.

4.1.2 Network Topologies. We consider three types of network topologies: deterministic, random and spatial networks. To avoid ambiguity we define each of the graphs as follows. We first define the following deterministic graphs with fixed degrees that we use in our simulations.

Line Graph-*L(n)*

A Line graph L(n) is a graph whose vertices $v_1, v_2, \cdots, v_n$ are connected with edges (v_i, v_{i+1}), $.i = 1, 2, \cdots, n-1$.

Ring Graph-*R(n)*

The ring graph R(n) on n vertices can be viewed as having a vertex set $\{0, 1, \cdots, n-1\}$ corresponding to the integers modulo n with edges $(i, i+1)$, modulo n.

We consider three random graphs in the simulation: Erdős-Rényi (ER), Random Regular (RR) and Linear Preference (LP) which we define as follows.

Linear Preference Graph- *LP* (n, α):

An LP (n, α) graph (also known as Barabasi Albert Graph) consists of n nodes is grown by adding new nodes each with α edges attached to existing nodes with probability proportional to the node degree. This has been shown to yield a power-law degree distribution [4].

Random Regular Graph- *RR* (n, β):

A β-regular graph RR (n, β) sampled from the probability space of all β-regular graphs on n vertices uniformly at random with $n\beta$ being even. For $\beta \geq 3$, a random β-regular graph of large size is asymptotically almost surely β-connected [7]. In our simulations, we assume $\beta \geq 3$.

Erdős-Rényi Graph- *ER* (n, γ):

The ER (n, γ) graph is generated by choosing each of the $[n(n-1)]/2$ possible edges with probability γ. $\gamma = \log n/n$ is a sharp threshold for the connectedness of $ER(n, \gamma)$. Also as $n \longrightarrow \infty$, the probability that $ER(n, \gamma)$ with $\gamma = 2\log n/n$ is connected, tends to 1 [7]. In all of our simulations, we assume $\gamma \geq \log n/n$.

We also evaluate the performance of proximity aware POT policies for three spatial graphs: Random Geometric (RG), Spatial Line (SL) and Spatial Ring (SR) graphs defined as follows.

2-D Random Geometric Graph-*RG* (n, r)

A 2-D random geometric graph RG (n, r) is an undirected graph with n nodes uniformly sampled from a 2-dimensional Euclidean space $[0, 1)^2$. Two vertices: $a, b \in V$ share an edge iff the Euclidean distance between these two servers is less than r, excluding any loops. RG (n, r) possesses a sharp threshold for connectivity at $r \sim \sqrt{\log n/\pi n}$ [14]. In all our simulations we consider $r \geq \sqrt{\log n/\pi n}$.

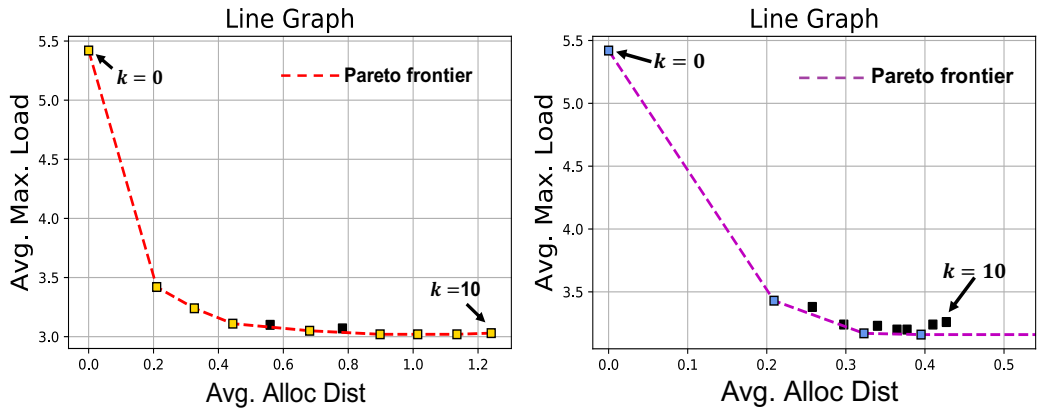
Spatial Line Graph-*SL*(n, L_{max})

Locations of servers are uniformly sampled from a one-dimensional euclidean space $[0, L_{max})$.

Spatial Ring Graph-*SR*(n, R)

We assume servers are placed uniformly at random on a circle of radius R.

We present the system and network parameters used in the simulation in Table 1. For random graphs, we assume the topology is re-sampled after each simulation run. Note that, the radius parameters for RG are chosen such that the graph remains asymptotically almost surely connected.



(a) (b)

Fig. 8. Pareto Frontier for multi-objective optimization between average maximum load and average allocation distance for servers on a Line graph with $m = n = 1000$ for (a) Unif-POT(k) and (b) InvSq-POT(k).

4.2 Maximum Load - Allocation Distance Tradeoff and Pareto Frontier

Similar to Section 3.3 we first discuss the inherent tradeoff between average maximum load and average allocation distance for different values of k in Unif-POT(k) and InvSq-POT(k) policies. We perform a simulation experiment with $n = 1000$ servers connected through a line graph. We assume $m = 1000$ jobs each arriving sequentially to one of the servers chosen uniformly at random and is allocated to a server according to Unif-POT(k) and InvSq-POT(k) policy.

We obtain both average maximum load and average allocation distance as a function of neighborhood parameter k. We observe that with an increase in the value of k the average maximum load value decreases for both policies. This is because the size of k-hop neighborhood of an origin server increases as k increases. Thus the load is distributed among a larger group of servers and the behavior of both policies resemble more and more that of POT policy for large values of k.

However, an increase in k results in larger values of average allocation distance. Again, for smaller values of k, the sampled servers remain close to the origin server. However, as k increases, the size of the k-hop neighborhood grows. One is more likely to sample a far away server thereby increasing the average allocation distance. Thus one needs to be careful in choosing the correct value of k according to the performance metric of interest.

Choosing the correct value of k, that minimizes both average maximum load and average allocation distance for a policy π, can be cast as a multi-objective optimization problem [13] as follows.

$$\begin{aligned} &\min\ (ML^{\pi}_{avg}(k), RD^{\pi}(k)), \\ s.t.\quad &k \in \{0, 1, 2, \cdots, k_{max}\}, \end{aligned} \tag{14}$$

where ML^{π}_{avg} and RD^{π} are the average maximum load and average allocation distance respectively. Here, k_{max} is the diameter of the network that connects all the servers. We solve the optimization problem (14) for the case when $\pi \in \{\text{Unif-POT}(k)\ ,\ \text{InvSq-POT}(k)\}$. We call a solution Pareto optimal/nondominated if none of the objective functions can be improved without degrading the other. Note that, since the objective functions in (14) are conflicting, there exists multiple Pareto optimal solutions. We call the set of Pareto optimal solutions as the Pareto frontier. In Figures 8 (a) and (b), we show the Pareto frontiers for policies Unif-POT(k) and InvSq-POT(k) respectively. In

Figure 8 (a)/(b), the entire red/pink dotted curve is the Pareto optimal solution, i.e., the optimal values of $k \in \{1, 2, \cdots, 10\} \setminus \{4, 6\}$ for Figure 8 (a) and $k \in \{0, 1, 4, 8\}$ for Figure 8 (b). All values of k that are on the Pareto frontier can be considered equally good with respect to the objective functions defined in (14).

4.3 Performance Comparison for Deterministic Graphs

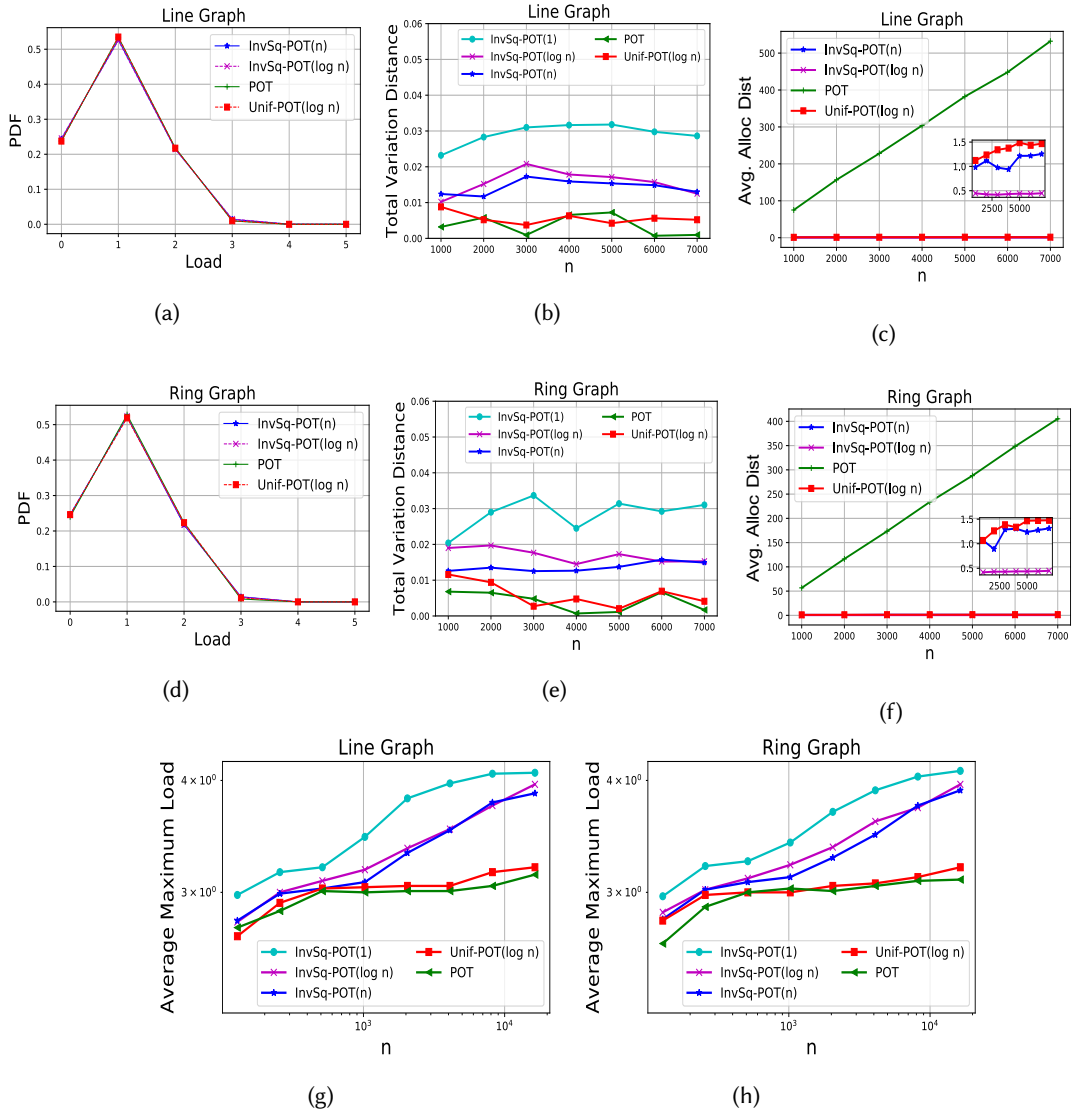


Fig. 9. Simulation Results for Unif-POT(k) and InvSq-POT(k) for line and ring graphs.

In this section, we analyze the performance of fixed degree deterministic graphs: Line and Ring. The results are presented in Figure 9. First we plot the load distribution for Line and Ring topologies

k	InvSq-POT(k)	Unif-POT(k)
1	0.03152	0.03054
2	0.02096	0.01562
4	0.01609	0.00703
8	0.01400	0.00359
16	0.01388	0.00144
10000	0.01272	0.00171

Table 2. Effect of k on the total variation distance of InvSq-POT(k) and Unif-POT(k) policies for a line graph with $n = 10000$.

in Figures 9 (a) and (d). Surprisingly, the load distributions of Unif-POT($\log n$), InvSq-POT($\log n$) and InvSq-POT(n) almost exactly match to that of POT for both Line and Ring topologies. Similar behavior is also observed in Figure 6 (a) for the case of planar graphs. We also consider the effect of k on the total variation distance as shown in the Table 2. We observe that the total variation distances decrease with k for both policies InvSq-POT(k) and Unif-POT(k). In [9], it was proved that in a $d(n)$-regular graph with dynamic setting, if a job is assigned to the server with the shortest queue among source server and a sampled server chosen from $d(n)$-neighbors of the source server uniformly at random, then the resulting load distribution is same as that of POT policy if $d(n) = \log(n)$ and n approaches infinity. We conjecture that this type of result holds even for our static models as observed numerically in Figures 6 (a), 9 (a) and (d). However, note that, the diameter of $d(n)$-regular graph is different than a line/ring topology which may impact the allocation distance even when the set of servers to sample from are of the same order in both cases.

Next, we compare the proximity based policies to POT with respect to total variation distance and present graphs for both Line and Ring topologies. We also plot the total variation distance between load distributions of two independent runs of POT which quantifies the noise or variation in load distribution of POT due to randomness. Again to our surprise, all proximity based policies with $k = \log n, n$ achieve total variation distances as low as 0.02 across a wide range of values of n. Also note that, InvSq-POT(1) achieves a load distribution farthest from POT while Unif-POT($\log n$) achieves the closest. Due to its uniform way of sampling, Unif-POT($\log n$) achieves the smallest total variation distance. However, due to bias towards closest severs, both InvSq-POT($\log n$) and InvSq-POT(n) achieve higher variation distances. Due to load-implementation cost trade-off, a very local policy InvSq-POT(1), achieves even higher variation distance. Both InvSq-POT($\log n$) and InvSq-POT(n) appear to converge to a constant variation distance as n gets large. We get similar results for the case when servers are connected through a ring graph. The results are presented in Figure 9 (e).

We plot average allocation distance as a function of number of servers in Figures 9 (c) and (f). The average path length can be thought of as an upper bound on average allocation distance under POT. Larger values of average path length imply larger values of average allocation distance under POT. Surprisingly, proximity based policies significantly decrease average allocation distance ($\sim$ 99% reduction) for large values of n. Since average path length for both line and ring graphs scale as $O(n)$, the average allocation distance under POT also drastically increases as n increases. Also note that, InvSq-POT($\log n$) achieves the lowest average allocation distance compared to other policies.

Finally Figures 9 (g) and (h) show the growth in average maximum load as n increases. We observe that both Unif-POT($\log n$) and POT produces the smallest average maximum load. The average maximum load of InvSq-POT($\log n$) and InvSq-POT(n) are similar but larger compared to

Unif-POT($\log n$). As expected InvSq-POT(1) exhibits the largest average maximum load since the policy distributes load only among immediate neighbors of the origin server.

4.4 Performance Comparison for Random Graphs

We now study the impact of proximity based policies on random graphs. The results are presented in Figure 10. We first plot the total variation distance between load distributions of proximity based policies and POT as a function of ER edge probability parameter γ as shown in Figure 10 (a). Note that, for all values of $k = 2, \log n, n$, both proximity based policies produce a variation distance as low as 0.5%. This is surprising since with $k = O(1) = 2$ we only sample the two hop neighborhood of the origin server. But we are able to produce load distribution behavior almost identical to that of POT, which samples from the entire set of servers. We observe similar trends for RR graphs, as shown in Figure 10 (d). However, we observe different results for the LP graph as shown in Figure 10 (g). For LP graphs, we observe that when $k = 2$, both proximity based policies produce higher total variation distances than when $k = \log n, n$. The variation distance is still small when $k = 2$ fluctuating around 0.033. One possible explanation for higher variation distances for small k can be the inhomogeneity of the degree distribution and the presence of hubs in a LP graph.

Note that, an increase in k should decrease variation distance since the sampling set size increases with k. Observe that the variation distance of policies under a Line or Ring topology is larger than that of any random topologies with fixed degree (Ex: RR topology). Higher graph densities in random topologies yield lower variation distance compared to Line or Ring topologies.

We now study the effect of network parameter on the average allocation distances of the proximity based policies as shown in Figures 10 (b), (e), (h). First observe that an increase in the value of network parameters (α, β and γ) increases the graph densities of the corresponding graphs (LP, RR and ER) and hence connectedness. This results in decrease in average allocation distances. Also observe the insensitivity of proximity based policies with $k = 2$ to the network size n. As expected, proximity policies with $k = 2$ produce very small allocation distances when compared to the case $k = \log n$.

Next, we study the scalability of average allocation distance with respect to network size as shown in Figures 10 (c), (f) and (i). Note that the average path length of ER and LP exhibits small ($\log n$) and ultra small world ($\log n / \log \log n$) behavior respectively [16]. Due to small world behavior, the observed average allocation distances are small for LP and ER topologies across all policies as compared to similar size Line and Ring topologies. Another implication of the small world behavior is that the $\log(n)$-hop and n-hop neighborhoods of a server are basically the same which results in very similar performance of InvSq-Pot($\log n$) and InvSq-POT(n) in Figures 10 (c), (f) and (i). Again as expected, proximity policies with $k = 2$ are insensitive to changes in network size and produce the smallest average allocation distances. Also, observe that between Unif-POT(k) and InvSq-POT(k) for every k, InvSq-POT(k) policies produce smaller average allocation distances for similar size networks.

We finally study how average maximum load increases as a function of network size as shown in Figures 10 (j), (k) and (l). For ER, for $k = 2, \log n, n$, both proximity based policies produce similar average maximum load values. For RR and LP we observe that all policies produce similar average maximum loads. However as the network size increases, local policies InvSq-POT(2) and Unif-POT(2) produce larger maximum load values compared to InvSq-POT(n), Unif-POT($\log n$), and POT.

4.5 Performance Comparison for Spatial Graphs

We first plot total variation distance as a function of radial parameter r of the RG topology as shown in Figure 11 (a). Again to our surprise, for all values of $k = \log n, n$, both the proximity based

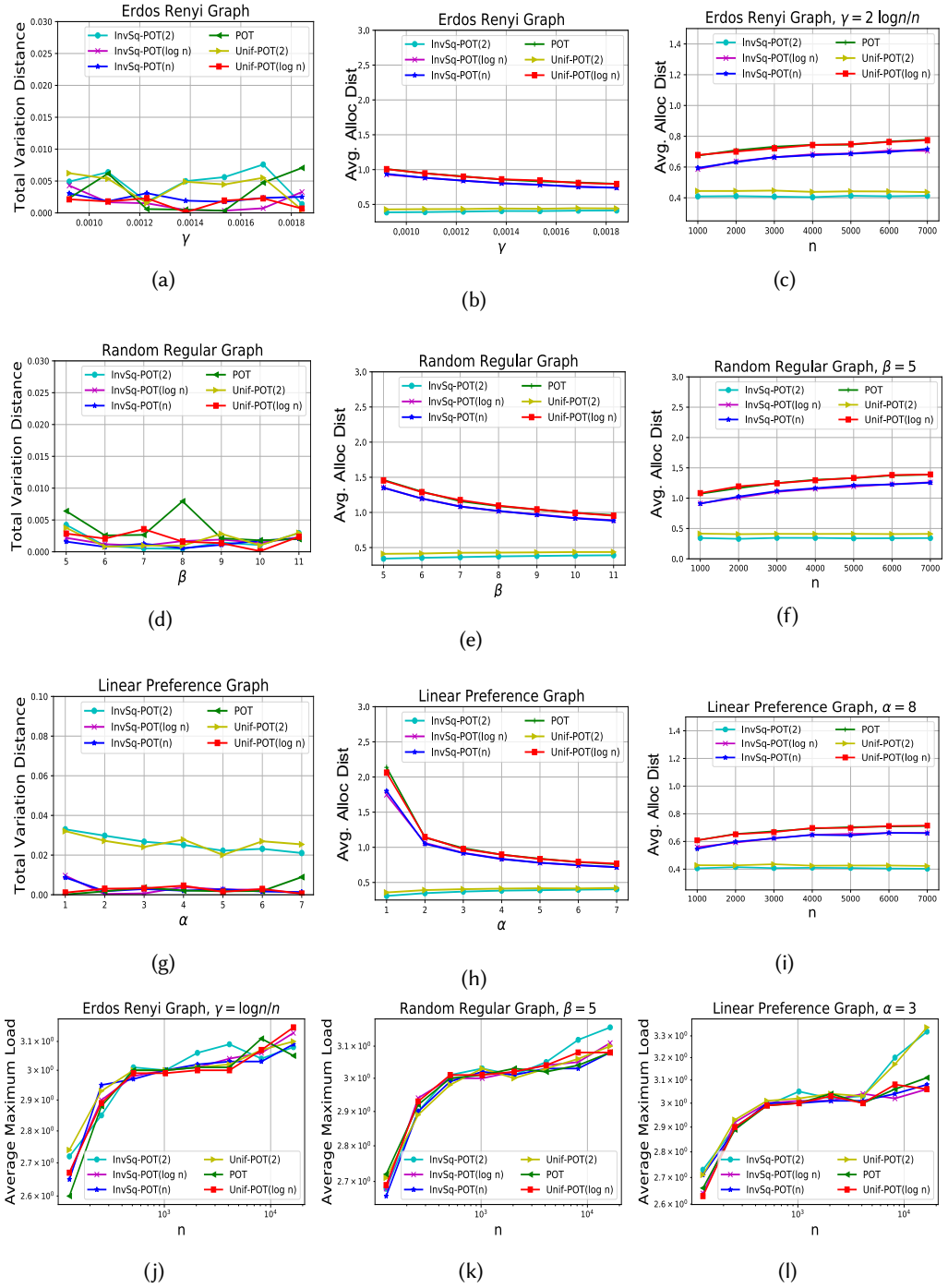


Fig. 10. Simulation Results for Unif-POT(k) and InvSq-POT(k) with $n = 10000$ and $k = 2, \log n, n$ for random graphs.

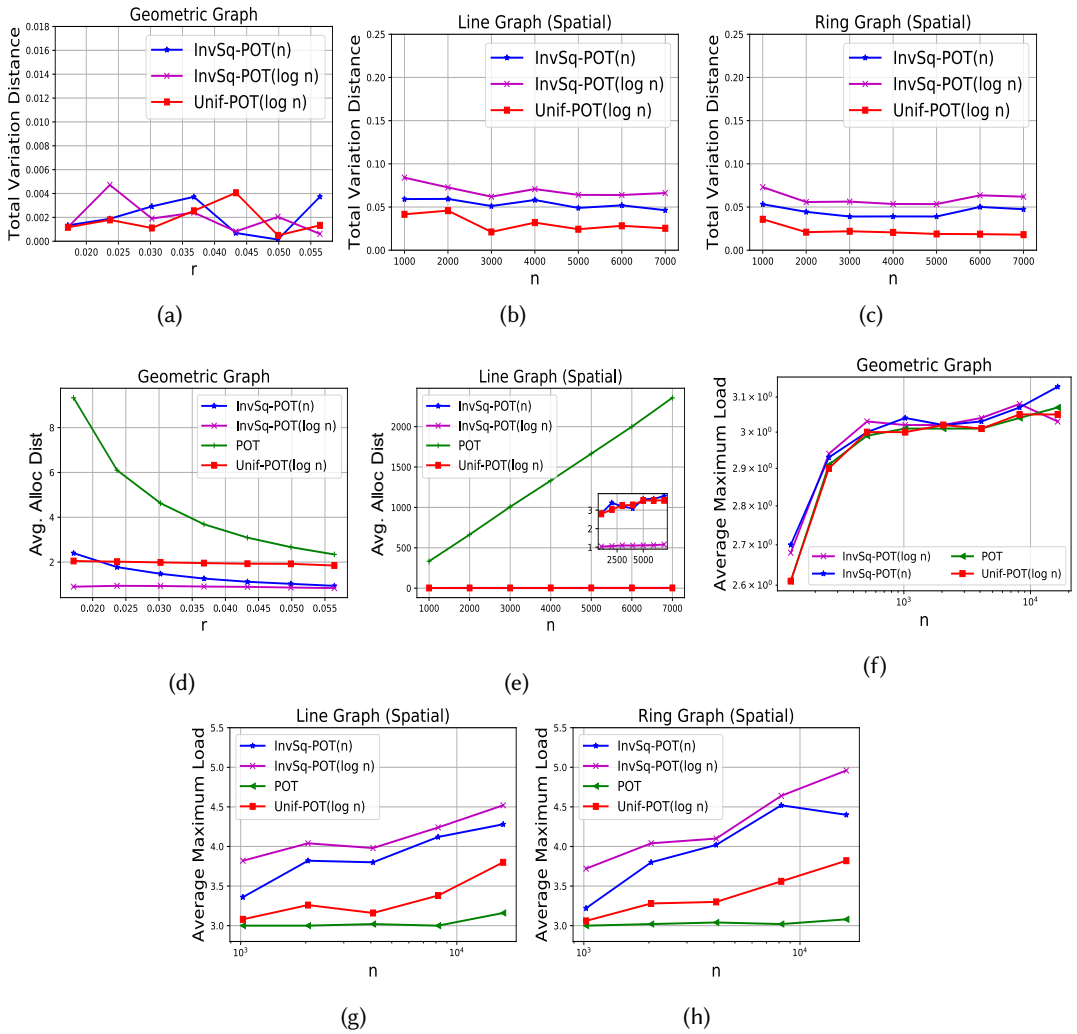


Fig. 11. Simulation Results for Unif-POT(k) and InvSq-POT(k) with $n = 10000$ and $k = \log n, n$ for spatial graphs.

policies produce a variation distance as low as 0.006. Note that for SL and SR topologies, we adopt a different job arrival model to incorporate the spatial nature of job request pattern. To be precise we assume both jobs and servers are placed uniformly at random on a one dimensional line $[0, L_{max})$ and on a circle of radius R for SL and SR topologies respectively. We plot variation distance as a function of network size for SL and SR as shown in Figures 11 (b), (c). We observe a clear trend of Unif-POT($\log n$) and InvSq-POT($\log n$) policy producing the smallest and largest variation distances for both SL and SR with InvSq-POT($\log n$) producing a variation distance of around 0.08. These variation distances are insensitive to network size. Also, note that, with introduction of the spatial dimension, the variation distances increased by five fold compared to their non-spatial counterparts (Figures 9 (b) and (e)) for the same network size.

We next plot average allocation distance as a function of r for RG topology as shown in Figure 11(d). First note that the proximity aware policies are almost insensitive to r. As r increases, the

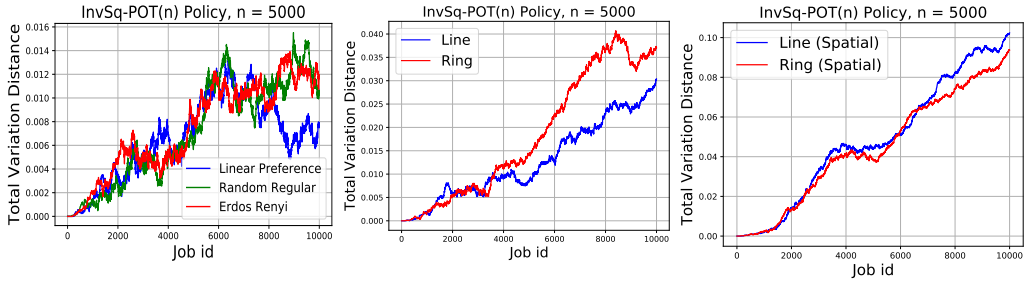


Fig. 12. Simulation Results for evolution of total variation distance for InvSq-POT(n) policy.

graph density for an RG increases thereby reducing average path length of the network. Thus we observe a decrease in average allocation distance for POT with an increase in r. As observed before, InvSq-POT(k) produces lower average allocation distances as compared to their Unif-POT(k) counterpart. InvSq-POT(k) produces the smallest average allocation distance for SL, which is almost insensitive to system size as shown in Figure 11 (e). However, as expected, POT produces a large allocation distance that increases linearly with system size.

We study the average maximum load behavior of the policies across various spatial networks in Figures 11 (f), (g), (h). For low values of n for RG topology, Unif-POT($\log n$) and POT exhibit similar average maximum loads. However, the corresponding average maximum load values for InvSq-POT($\log n$) and InvSq-POT(n) are larger. When n is large, all policies exhibit similar behavior. For SL and SR topologies, InvSq-POT($\log n$) is the worst while POT is the best policy. Unif-POT($\log n$) is closer to POT while InvSq-POT(n) is closer to InvSq-POT($\log n$) with respect to average maximum load.

4.6 Evolution of Total Variation Distance

Until now we have considered systems where there are an equal number of servers and users/jobs, i.e. $m = n$. In this section we study the evolution of the total variation distance for a fixed n and the underlying network topology. We evaluate the load distribution of POT and other proximity aware policies after each job arrival. We then calculate the total variation distance after each job arrival and plot its evolution as shown in Figures 12 (a), (b), (c). While we only focus on InvSq-POT(n) policy for Figures 12 (a)-(c), we obtained similar results for other proximity aware policies.

We consider a network of $n = 5000$ servers. We observe the system from arrival of first job till the 10000^{th} job. We plot the evolution of variation distance for various random network topologies as shown in Figure 12 (a). We set the parameters for all three random graphs: ER, LP and RR such that their graph density remains almost equal. To be precise we set $\alpha = \log n, \beta = 2\log n$ and $\gamma = 2\log n/n$. First we observe that with increase in number of jobs the total variation distance continuously increases for RR and ER networks. However, for this particular choice of network parameters, variation distance for LP first increases and then decreases. Thus one can believe that proximity aware policies on scale free networks may provide good load balancing properties when there is imbalance between the number of servers and number of jobs. However, no such phenomena is observed for non-spatial and spatial Line and Ring topologies.

5 RELATED LITERATURE

The randomized load balancing problem can be categorized into two versions: static and dynamic. This work mainly deals with designing static load balancing policies while preliminary results on dynamic load balancing policies were also presented in Section 3.5. For completion purpose, below we discuss the state-of-the-art related to both static and dynamic version.

5.1 Static Load Balancing

Many previous works [1, 25] have developed simple and efficient load balancing algorithms in the static setting. The widely acclaimed *Power of Two* (POT) choices policy was first proposed by Azar et al. [3]. Further generalizations of POT policy to account for correlated and non-uniform sampling strategies have been discussed in subsequent works [6, 10, 35]. Load balancing on graphs was first proposed by Kenthapadi et al. [22], where the authors considered a model with bins interconnected as a Δ-regular graph. Each ball then samples a random edge of the graph and gets allocated at one of its endpoints with smaller load. Godfrey et al. [18] generalized the graph based model to balanced allocations on a hypergraph. Bringmann et al. [8] studied a model where each ball picks a random bin and performs a local search from the bin to a bin with local minimum load and gets allocated to it. Pourmiri et al. [29] proposed algorithms for allocating balls to bins that are interconnected as a regular graph by performing a non-backtracking random walk from a chosen node. In [28], authors showed that the gap between the load of the most loaded bin and the average in a graph setting depends on the edge expansion of the graph connecting the bins. Authors in [30] consider load balancing in a network of caching servers that deliver contents to end users. In [11], parallel load balancing protocols were designed for a client-server distributed model where client set and servers were connected to each other via a fixed bipartite graph. Tang et.al. [33] developed two new policies to allocate n balls into n bins by non-backtracking random walk on a k-regular graph with the bins being the vertices of the graph. Balanced balls to bins allocation for the case when bins are represented as dynamic hypergraphs, is discussed in [19].

5.2 Dynamic Load Balancing

In the dynamic setting, it was shown in [25, 36] that under the Power of d ($d \geq 2$) policy, the stationary probability that a server has at least i progressing jobs in the asymptotic regime when $n \to \infty$ is equal to $(\lambda/\mu)^{(d^i-1)/(d-1)}$, whereas it equals to $(\lambda/\mu)^i$ when $d = 1$. This shows that the POT policy reduces the average delay significantly. In [17], a load balancing policy was investigated for symmetric graphs. There, a job that arrived at a server say s was served at the shortest queue length server among servers s and m, where m was picked uniformly at random from the set of neighboring servers of s. The authors were able to derive a set of evolution equations based on pair-wise approximations and the fixed-point of these equations was shown to approximate the stationary distribution of a server's state. In [9], a similar model in which servers are located at the nodes of a deterministic graph G_n was studied. The authors showed that if $d_{min}(G_n) \to \infty$ and $\sup_{i\geq 1} \left|[d_{min}(C_{i,n})/d_{max}(C_{i,n})] - 1\right| \to 0$, where $d_{min}(G_n)$ indicates the minimum degree of G_n, $C_{i,n}$ is a connected component of the graph and $d_{max}(C_{i,n})$ denotes the maximum degree of a node in $C_{i,n}$, then the empirical process of occupancy converges to the same mean-field as in the case of POT policy. They also showed that for Erdős-Rényi graphs with average degree D_N, the empirical process of occupancy converges to the same mean-field limit as in the POT policy if $D_n/\ln(n) \to \infty$ as $n \to \infty$. Recently, in [32], a load balancing policy was studied for a bipartite graph in which task types are matched to servers which can serve them and each task type can be processed only at a small subset of servers. An incoming task is assigned to a server that has the shortest queue size among d randomly chosen servers from the set of servers which can process it. Under the assumption that if a graph satisfies certain connectivity properties referred to as proportional

sparsity, they showed that empirical occupancy process converges to the same mean-field limit as in the case of a complete bipartite graph. For random graphs that satisfy proportional sparsity, if the degree of a server is at most D_n satisfying $D_n \to \infty$ and $nD_n/M(n)\ln(n) \to \infty$ as $n \to \infty$, then the empirical occupancy process was shown to converge to the same mean-field limit as in the complete bipartite graph case. Authors in [15] and [31] consider the analysis for greedy based user routing policies when servers are placed on a real line and circle respectively.

6 CONCLUSION

In this work we considered a class of proximity aware power of two choices based allocation policies where both servers and users are located on a two-dimensional plane. We analyzed the sPOT policy and provided expressions for the lower bound on the asymptotic maximum load on the resources. We claim that for both grid and uniform based resource placement, sPOT does not provide POT benefits. We proposed two non-uniform euclidean distance based server sampling policies that achieved the best load and allocation distance behavior. We also considered the case when servers are interconnected as an arbitrary graph. We performed extensive simulations over a wide range of network topologies. To our surprise, with few simple modifications in the server sampling process, we observed a drastic reduction in the overall system wide implementation cost while obtaining a similar load distribution profile as that of POT policy. Finally, going further, we aim at extending our results to consider dynamic load balancing systems on arbitrary graphs.

7 ACKNOWLEDGMENT

This research was sponsored by the U.S. ARL and the U.K. MoD under Agreement Number W911NF-16-3-0001 and by the NSF under Grant CNS-1617437. This document does not contain technology or technical data controlled under either the U.S. International Traffic in Arms Regulations or the U.S. Export Administration Regulations. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation, U.S. ARL or the U.K. MoD.

REFERENCES

[1] M. Adler, S. Chakrabarti, M. Mitzenmacher, and L. Rasmussen. Parallel randomized load balancing. *Random Structures and Algorithms*, 13(2):159–188, 1998.
[2] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A Survey. *Computer Networks*, 54(15):2787–2805, 2010.
[3] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, 1999.
[4] A. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.
[5] B. A. Bash and P. J. Desnoyers. Exact Distributed Voronoi Cell Computation in Sensor Networks. In *IPSN*, 2007.
[6] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking. Balanced allocations: The heavily loaded case. *SIAM Journal on Computing*, 35(6):1350–1385, 2006.
[7] B. Bollobas. *Random Graphs*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2 edition, 2001.
[8] K. Bringmann, T. Sauerwald, A. Stauffer, and H. Sun. Balls into bins via local search: Cover time and maximum load. *Random Structures and Algorithms*, 48(4):681–702, 2016.
[9] A. Budhiraja, D. Mukherjee, and R. Wu. Supermarket model on graphs. *Ann. Appl. Probab.*, 29(3):1740–1777, 2019.
[10] J. Byers, J. Considine, and M. Mitzenmacher. Geometric Generalizations of the Power of Two Choices. In *SPAA*, 2004.
[11] A. Clementi, E. Natale, and I. Ziccardi. Parallel Load Balancing on Constrained Client-Server Topologies. *Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 163–173, 2020.
[12] C. Cooper, M. Dyer, and C. Greenhill. Sampling regular graphs and a peer-to-peer network. *Combinatorics Probability and Computing*, 16(4):557–593, 2007.
[13] K. Deb. Multi-objective Optimization BT - Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques. pages 403–449. Springer US, Boston, MA, 2014.
[14] J. Diaz, D. Mitsche, and X. Perez-Gimenez. On the connectivity of dynamic random geometric graphs. *SODA*, pages 601–610, 2008.

[15] S. Foss, L. T. Rolla, and V. Sidoravicius. Greedy walk on the real line. *Annals of Probability*, 43(3):1399–1418, 2015.
[16] A. Fronczak, P. Fronczak, and J. A. Hołyst. Average path length in random networks. *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, 70(5):7, 2004.
[17] N. Gast. The power of two choices on graphs: the pair-approximation is accurate. In *In Proc. MAMA workshop 2015*, pages 69–71, 2015.
[18] P. B. Godfrey. Balls and bins with structure: Balanced allocations on hypergraphs. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 511–517, 2008.
[19] C. Greenhill, B. Mans, and A. Pourmiri. Balanced allocation on dynamic hypergraphs. *Leibniz International Proceedings in Informatics, LIPIcs*, 176, 2020.
[20] K. K. and R. Panigrahy. Balanced Allocation on Graphs. In *SODA*, 2006.
[21] R. Kacimi, R. Dhaou, and A. L. Beylot. Load balancing techniques for lifetime maximizing in wireless sensor networks. *Ad Hoc Networks*, 11(8):2172–2186, 2013.
[22] K. Kenthapadi and R. Panigrahy. Balanced Allocation on Graphs. 2005.
[23] J. M. Kleinberg. Navigation in a small world. *Nature*, 406(6798):845, 2000.
[24] A. W. Marshall and I. Olkin. Inequalities: Theory of Majorization and its Applications. In *In: Academic Press*, 1979.
[25] M. D. Mitzenmacher. The Power of Two Choices in Randomized Load Balancing. In *Ph.D. Dissertation, Harvard University*, 1996.
[26] A. Okabe, B. Boots, and K. Sugihara. Spatial Tessellations Concepts and Applications of Voronoi Diagrams. In *New York: Wiley*, 1992.
[27] M. Penrose. *Random Geometric Graphs*. 2007.
[28] Y. Peres, K. Talwar, and U. Wieder. Graphical balanced allocations and the $(1 + \beta)$-choice process. *Random Structures and Algorithms*, 47(4):760–775, 2015.
[29] A. Pourmiri. Balanced allocation on graphs: A random walk approach. *Random Structures and Algorithms*, 55(4):980–1009, 2019.
[30] A. Pourmiri, M. J. Siavoshani, and S. P. Shariatpanahi. Proximity-Aware Balanced Allocations in Cache Networks. *Proceedings - 2017 IEEE 31st International Parallel and Distributed Processing Symposium, IPDPS 2017*, pages 1068–1077, 2017.
[31] L. T. Rolla and V. Sidoravicius. Stability of the Greedy Algorithm on the Circle. *Communications on Pure and Applied Mathematics*, 70(10):1961–1986, 2017.
[32] D. Rutten and D. Mukherjee. Load Balancing Under Strict Compatibility Constraints. In *arXiv:2008.07562*, 2020.
[33] D. Tang and V. G. Subramanian. Balanced Allocation on Graphs with Random Walk Based Sampling. *2018 56th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2018*, pages 765–766, 2019.
[34] S. R. Turner. The effect of increasing routing choice on resource pooling. *Probability in the Engineering and Informational Sciences*, 12(1):109–124, 1998.
[35] B. Vöcking. How asymmetry helps load balancing. *Journal of the ACM*, 50(4):568–589, 2003.
[36] N. D. Vvedenskaya, R. L. Dobrushin, and F. I. Karpelevich. Queueing system with selection of the shortest of two queues: An asymptotic approach. *Problemy Peredachi Informatsii*, 32(1):20–34, 1996.