Abstract—The map-reduce framework is used in many data-intensive parallel processing systems. Data locality is an important problem with map-reduce as tasks with local data complete faster than those with remote data. We propose a degree-guided task assignment algorithm, which uses very little extra information than the currently implemented Random Server algorithm. We analyze a simple version of the degree-guided algorithm, called Peeling algorithm, and the Random Server algorithm in a discrete-time model using evolution of random graphs. We characterize the thresholds below which no queueing takes place and compute the effective service rates for both algorithms. The degree-guided algorithm achieves the optimal performance in the region of practical interest and significantly outperforms the Random Server algorithm. The performance characteristics derived from discrete time model are confirmed with simulation in continuous time.

I. INTRODUCTION

Cluster computing systems, such as MapReduce [1] and Hadoop [2], have become a popular framework for data-intensive applications. Large clusters consisting of tens of thousands of machines [3] have been built for web indexing and searching; small and mid-size clusters have also been built for business analytics and corporate data warehousing [4]. In clusters of all sizes, throughput and job completion time are important metrics for computation efficiency.

To facilitate parallel computation, a file is divided into many small chunks. Each chunk is replicated three times by default and placed on different servers. The default number is determined based on availability concerns [5]: The first chunk is placed on a randomly chosen server; the second chunk is placed on a random server in the same rack to protect against server failures, and the third chunk is placed on a random server outside the rack to protect against rack failures. When a task is placed on a server where data are not locally available, it will need to retrieve data from one of the remote servers hosting the replicas. This increases the completion time of the task. As a result, placing tasks as close as possible to data is a common practice of data-intensive systems, referred to as the data locality problem [1].

Data locality is an important problem as it significantly affects system throughput and job completion times. The current scheduling algorithms in Hadoop are based on the Random Server algorithm [6], which depends on a large number of outstanding tasks to achieve high data locality. As a result, with light to medium load, which is the region online systems operate in today, the Random Server algorithm results in unnecessary delay of tasks.

In this paper, we propose a degree-guided task assignment algorithm that significantly outperforms the Random Server algorithm at light to medium load. We analyze the degree-guided algorithm and the Random Server algorithm in a discrete-time model using evolution of random graphs. We show that the degree-guided algorithm has the same performance as the optimal maximum matching algorithm below a threshold, and converges to that of the Random Server algorithm at high load. We characterize the thresholds below which no queuing takes place for both algorithms. The degree-guided algorithm experiences queueing at a higher load than the Random Server algorithm and has a significantly higher effective service rate in the region of light to medium load. The performance characteristics are further verified with simulation in continuous time.

This paper is organized as follows. In Section II, we describe the algorithms and the model used for analysis. Section III derives the mean-field model of the assignment procedure for a single time slot. We show the performance of the Random Server algorithm and the Peeling algorithm over one time slot in Section IV. Section V characterizes the fixed points and the thresholds below which no queuing takes place for both algorithms. We evaluate these two algorithms via simulation in continuous time in Section VI.

II. ALGORITHM DEFINITION

A. Discrete-time Model

Consider a discrete-time model for a system with $m$ parallel servers. At the beginning of each time slot, a constant number of tasks arrive at the system. Each task processes one file chunk, which is replicated $d$ times and placed on $d$ randomly selected servers.

The placement of data can be modelled by bipartite graphs $G$ with $n$ task nodes and $m$ server nodes, as illustrated in Fig. 1. An edge between task node $i$ and server node $j$ indicates the presence of data for task $i$ on server $j$. And we define the degree of a server node as the number of unassigned tasks that have data on it. In particular, we consider the default data replication scheme of 3 replicas for each data chunk, i.e., $d = 3$. The scheme can be easily extended to a variable number of replicas for different data chunks, as proposed in [3], [7].
Assume that each server can process only one task at any time. Task assignment is performed at the beginning of a slot. All tasks are assigned unless there are no more idle servers, and the remaining tasks are kept in a queue. For a task assigned to a server with its data, the service time is assumed to follow geometric distribution with parameter $p$, and the parameter is $q$ for tasks processed at servers without its data, where $q < p$. We refer to servers serving tasks with local data as $p$-server and servers serving tasks without local data as $q$-server.

### B. Algorithm Description

We consider the following two algorithms.

**1. Random Server algorithm:** Whenever there are outstanding tasks in the system, an idle server is chosen randomly. If there exist multiple tasks whose data is replicated on this server, one task is uniformly selected from the set. Otherwise, this server is assigned for a randomly selected task. This models the FIFO scheduler currently used in Hadoop clusters [2], [7], which assigns tasks following exactly the same rule.

**2. Degree-guided algorithm:** An idle server with the least non-zero degree is sampled if outstanding tasks are present. And this server is assigned for a task that’s randomly selected from all of its connected tasks. When all idle servers are of degree 0, a server is selected randomly for a random task. The least non-zero degree of the selected server ensures that the task is assigned to a p-server. In addition, it maintains the connections of unassigned tasks to the remaining idle servers to the utmost extend, which increases the probability of assigning these tasks to p-servers.

In this paper, we consider a simple version of the degree-guided algorithm, called **Peeling algorithm.** With the Peeling algorithm, an idle server that has local data for only one outstanding task is assigned to this task. The procedure continues until no server of degree 1 exists, which is referred to as the peeling stage. Then the assignment procedure continues with the Random Server algorithm, which is called the random stage. Note that the Peeling algorithm is equivalent to the degree-guided algorithm until the peeling stage stops.

### III. MEAN-FIELD MODEL FOR A SINGLE TIME SLOT

We need the following definitions to derive the mean-field models for the Random Server algorithm and the Peeling algorithm. In this section, we focus on the analysis of these two algorithms over one time slot.

**Definition 1:** Degree Distributions from A Node Perspective. Given a graph $G$ with $n$ task nodes and $m$ server nodes, let $L_i$ denote the number of task nodes of degree $i$, $i = 0, 1, \cdots, l_{\text{max}}$, and $R_j$ the number of server nodes of degree $j$, $j = 0, 1, \cdots, r_{\text{max}}$, where $l_{\text{max}}$ and $r_{\text{max}}$ are the largest degree of task nodes and server nodes respectively. So $\sum_i L_i = n$ and $\sum_j R_j = m$. The degree distributions from a node perspective are defined by the pair $(L, R)$, where $L = \{L_0, L_1, \cdots, L_{l_{\text{max}}}\}$ and $R = \{R_0, R_1, \cdots, R_{r_{\text{max}}}\}$.

**Definition 2:** The Standard Ensemble $G(L, R)$. Given the degree distributions $(L, R)$, we define an ensemble of bipartite graphs $G(L, R)$ in the following way. Each graph in $G(L, R)$ has the degree distributions $(L, R)$. As a node of degree $i$ has $i$ sockets from which the $i$ edges emanate, there are a total of $\sum_j i L_i = \sum_j j R_j$ sockets on each side. Label the sockets on each side with numbers $\{1, \cdots, \sum_i i L_i\}$. Let $\sigma$ be a permutation on $\{1, \cdots, \sum_i i L_i\}$, and associate $\sigma$ to a bipartite graph by connecting the $j$-th socket on the task nodes to the $\sigma(j)$-th socket on the server nodes. We define a probability distribution over the set of graphs by placing the uniform probability distribution on $\sigma$. The definition is the same as the standard ensemble for LDPC codes [8], pg 78.

We model the assignment procedure as an evolution of the random graph ensemble. Let $s \in \mathcal{N}$ denote the assignment step, which starts at zero and increases by one for each task assigned. At each step, all edges connected to the selected server and those connected to the assigned task are removed from the graph. This procedure results in a sequence of residual graphs, denoted by $G(L(s), R(s))$, where $(L(s), R(s))$ are the degree distributions of unserved tasks and remaining idle servers at step $s$. No edge is left at the end of the assignment as either all tasks are assigned or no idle server remains. Consider a system with $n$ unserved tasks and $k$ idle servers before the assignment $(k \leq m)$. Then $\frac{L_0(k)}{n}$ follows a binomial distribution $B(d, \frac{k}{m})$ and $\frac{R_0(k)}{k}$ a binomial distribution $B(dn, \frac{k}{m})$ truncated at $r_{\text{max}}$. Hence $l_{\text{max}} = d$ and we can set $r_{\text{max}}$ a fixed constant as there is a limited amount of storage space on each server. Let $M_p(s)$ and $M_q(s)$ denote the increased numbers of p-servers and q-servers at the end of step $s$ respectively. Define the scaled time $\tau = \frac{w}{m}$, where $w = nd$ denotes the total number of edges in the initial graph. Let $\gamma_i(\tau) = \frac{R_i(w)}{w}$, $l_i(\tau) = \frac{L_i(w)}{w}$, $m_p(\tau) = \frac{M_p(w)}{w}$, $m_q(\tau) = \frac{M_q(w)}{w}$, which together determine the assignment path for a single slot. We obtain the following theorem. Full proof can be found in [9].

**Theorem 1:** Evolution of Residual Graph for the Assignment Algorithms. The expected assignment paths of the two algorithms are described by the two sets of differential equations respectively: Random Server algorithm:

$$
\frac{dl_i(\tau)}{d\tau} = -\frac{l_i(\tau)}{v(\tau)e(\tau)} + \sum_{j \geq 1} \gamma_j (j - 1)(i + 1)l_{i+1} - jil_i,
$$

for $0 \leq i \leq l_{\text{max}} - 1$ (1)
\[
\frac{d\gamma_i(\tau)}{d\tau} = -\frac{\gamma_i}{c(\tau)} + \left[ (i + 1)\gamma_{i+1} - i\gamma_i \right] \frac{\gamma_0}{v(\tau)c(\tau)} \\
+ \frac{1}{c(\tau)} \sum_{k=1}^{s(s-1)} \frac{s(s-1)l_k}{c'(\tau)} \\
\text{for } 0 \leq i \leq r_{\text{max}} - 1
\]

\[
\frac{dm_p(\tau)}{d\tau} = 1 - \frac{\gamma_0}{c(\tau)} \quad \text{and} \quad \frac{dm_q(\tau)}{d\tau} = \frac{\gamma_0}{c(\tau)}
\]

where \( v(\tau) = \sum_j l_j(\tau) \), \( c(\tau) = \sum_j \gamma_j(\tau) \), and \( e(\tau) = \sum_j j\gamma_j(\tau) \).

Peeling algorithm:
\[
\frac{d\gamma_i(\tau)}{d\tau} = \frac{i\gamma_i}{\sum_j j\gamma_j(\tau)} \quad \text{for } 2 \leq i \leq l_{\text{max}} - 1,
\]

\[
\frac{d\gamma_i(\tau)}{d\tau} = (2\gamma_2 - \gamma_i) \sum_k (k-1)kl_k \sum_j j\gamma_j(\tau) - 1,
\]

\[
\frac{d\gamma_i(\tau)}{d\tau} = ((i+1)\gamma_{i+1} - i\gamma_i) \sum_k (k-1)kl_k \sum_j j\gamma_j(\tau) \quad \text{for } 2 \leq i \leq r_{\text{max}} - 1,
\]

\[
m_p(\tau) = \tau \quad \text{and} \quad m_q(\tau) = 0
\]

where \( v(\tau), c(\tau) \) and \( e(\tau) \) are defined as above.

With probability at least 1 - \( O(n \frac{\epsilon^2 e^{-\frac{(\epsilon^2)}{2}}} \), the assignment path of a specific instance has maximum \( L \)-distance from the expected assignment path at most \( O(n \epsilon^2 e^{-\frac{(\epsilon^2)}{2}}) \) from the start of the process until either the total number of nodes in the residual graph has reached size \( \eta n \), where \( \eta \) is an arbitrary strictly positive constant, or the algorithm gets stuck.

By solving these differential equations, we can obtain the expected increased fractions of p-servers and q-servers after the assignment, which provide an efficient way to evaluate the performance of these two algorithms.

IV. PERFORMANCE FOR A SINGLE TIME SLOT

Let \((f_i, f_p, f_q)\) denote the fraction of idle servers, p-servers and q-servers at the beginning of a time slot, hence \( f_i + f_p + f_q = 1 \). We first show the property of function \( \sigma_p(f_i, f_p) = \frac{M_p(n)}{m} \), which denotes the increased fraction of p-servers at the end of assignment process for a system with \( \frac{n}{m} = r \) and \( f_i \) percent of idle servers before the assignment. We then evaluate the performance of the two algorithms against the maximum matching, which is optimal.

A. Properties of the function \( \sigma_p(f_i, f_p) \)

First, we show that for the Random Server algorithm, the function \( \sigma_p(f_i, f_p) \) does not depend on the fraction of idle servers available at the beginning of the slot, so long as all tasks are assigned eventually.

**Theorem 2: Independence of \( f_i \).** Consider the Random Server algorithm for a system with \( m \) servers and \( n = mr \) unassigned tasks. Let the fraction of servers be \((f_i, f_p, f_q)\) before the assignment. If \( r < f_i \), the increased fraction of p-servers, \( \sigma_p(f_i, f_p) \), is independent of \( f_i \).

We sketch the proof of Theorem 2 here and full proof can be found in [9]. Note that the evolution of the assignment procedure only depends on the degree distribution of tasks nodes and servers nodes, and the sequence of idle servers sampled. The random selection of idle servers, it makes no difference to determine the sequence of idle servers sampled \( I_0 \) before the assignment. Then restrict the original graph to the tasks nodes and \( I_{n-1} \), denoted by \( G' \). With \( I_0 \) fixed, \( \sigma_p \) is determined by \( G' \), which has the same degree distributions for different \( f_i \). Hence the resulting \( \sigma_p \) is the same.

Figure 2 shows the plots of \( \sigma_p(r, f_i) \) for the Random Server algorithm and the Peeling algorithm with different fraction of idle servers, \( f_i \), at the beginning of the assignment process.

Note that for Peeling algorithm with \( f_i = 1 \), there exists an obvious point for \( r \), above which \( \sigma_p(r, f_i) \) decreases before increases again. Below the threshold, \( \gamma_1(\tau) > 0 \) for \( \tau \in [0, \min\{\frac{1}{2}, \frac{m}{2}\}] \). That is, servers of degree-1 are available throughout the assignment procedure so the peeling stage doesn’t stop. Hence \( \sigma_p(r, f_i) = \min\{r, f_i\} \), which equals \( r \) if \( r < f_i \). With \( f_i < 1 \), however, some tasks are only connected to occupied servers, which will result in the emergence of random stage and hence decrease \( \sigma_p \).

We also observe that \( \sigma_p(r) \) increases monotonically for the Random Server algorithm. We have the theorem below.

**Theorem 3: Monotonicity.** Consider the Random Server algorithm for a system with \( m \) servers and \( n = mr \) tasks arriving. Let \( f_i = 1 \) and \( r < f_i \). Then the increased fraction of p-servers, \( \sigma_p(r) \), strictly increases with \( r \).

The idea of proof for Theorem 3 is induction. We show that \( \sigma_p(0) < \sigma_p(mr) \) for \( n, m \in N \) using coupling. For detailed proof, please refer to [9].

B. Comparison with maximum matching

The objective of task assignment is to assign as many tasks as possible to a server with local data, which is a matching problem. The following theorem shows the performance of the maximum matching algorithm, which is optimal.
Theorem 4: [10] Consider a system where $n, m \to \infty$ with $r = \frac{n}{m}$ fixed and each task has its data on $d$ randomly selected servers. Let $y$ be the unique solution to the equation:
\[
d = \frac{y(1 - e^{-y})}{1 - e^{-y} - ye^{-y}} \quad (8)
\]
and $r^* = \frac{y}{d(1 - e^{-y})^2}$. If $r \leq r^*$, all tasks are assigned to p-servers; if $r \geq r^*$, proportion of p-servers assigned is:
\[
\sigma_p \to r + 1 - e^{-z} - ze^{-z} - \frac{z}{d(1 - e^{-z})} \quad (9)
\]
where $z = dx$ and $x$ is the largest solution to
\[
x = (1 - e^{-dx})d^{-1} \quad (10)
\]
For the case of $d = 3$, we obtain $r^* = 0.918$. The performance of maximum matching is showed in Fig. 2. We can see that with all servers idle initially, the Peeling algorithm achieves the optimal performance as maximum matching, if $r$ is below the threshold for the emergence of the random stage. And the improvement of the Peeling algorithm over the Random Server algorithm is significant. Above the threshold, the Peeling algorithm deteriorates from the optimal matching, but still performs better than the Random Server algorithm.

We have the following lemma.

Lemma 1: The Peeling algorithm achieves the performance of optimal matching when the load for the system is below the threshold for the emergence of the random stage.

V. FIXED POINT CHARACTERIZATION

In this section, we characterize the fixed points of the system evolution when the load is below the threshold that no task remains in the queue after the assignment.

Consider a system with $r = \frac{n}{m}$ fixed and $n, m \to \infty$. Let $(\pi_i, \pi_p, \pi_q)$ denote the equilibrium values of $(f_i, f_p, f_q)$ before the assignment without tasks queueing, and $(\hat{\pi}_i, \hat{\pi}_p, \hat{\pi}_q)$ the equilibrium values after the assignment.

Theorem 5: Fixed point characterization. For both the Random Server algorithm and the Peeling algorithm, define
\[
g(\pi_i) = 1 - \frac{\sigma_p(r, \pi_i)}{\sigma_q(r, \pi_i)} = \frac{r - \sigma_p(r, \pi_i)}{\sigma_q(r, \pi_i)}
\]
where $g(\pi_i)$ is different for these two algorithms. If no queueing takes place, $\pi_i$ satisfies
\[
\pi_i = g(\pi_i) + r \quad \text{and} \quad g(\pi_i) > 0. \quad (11)
\]
And the fixed point is
\[
(\pi_i, \pi_p, \pi_q) = \left( \frac{1}{\pi_i}, \frac{\sigma_p(r, \pi_i)}{\sigma_q(r, \pi_i)}, \frac{1 - q}{\pi_i} \right)
\]
\[
(\hat{\pi}_i, \hat{\pi}_p, \hat{\pi}_q) = \left( \frac{\pi_i}{\gamma}, \frac{\sigma_p(r, \pi_i)}{\sigma_q(r, \pi_i)}, \frac{r - \sigma_p(r, \pi_i)}{\sigma_q(r, \pi_i)} \right).
\]

Proof. With $n = mn$ and no tasks in the queue, at the fixed point, all $n$ tasks are assigned, which yields the following:
\[
\hat{\pi}_p = \pi_p + \sigma_p(r, \pi_i), \quad \hat{\pi}_q = \pi_q + r - \sigma_p(r, \pi_i), \quad \hat{\pi}_i = \pi_i - r.
\]
With the geometric distribution for the service time, a task leaves a p-server with probability $p$ and leaves a q-server with probability $q$, we have
\[
\pi_i = \pi_i + \hat{\pi}_p p + \hat{\pi}_q q, \quad \pi_p = \hat{\pi}_p (1 - p), \quad \pi_q = \pi_q (1 - q).
\]
Solving the equations, we obtain
\[
\pi_i = g(\pi_i) + r.
\]
To ensure all tasks assigned, we need $\pi_i > r$, which yields
\[
\pi_i \to \infty \quad (\text{Random Server} \text{ algorithm with } \pi_i > r).
\]

Remark. For the Random Server algorithm with $\pi_i > r$, Theorem 2 indicates that $\sigma_p(r, \pi_i)$ only depends on $r$. So the computation of its fixed point can be simplified.

A. Queuing threshold

From Theorem 5, we can define the threshold for queueing. Definition 3: Threshold for Queueing. We define the threshold of load below which no queueing takes place as
\[
\rho^* (p, q) = \sup \{ r \in [0, 1] : \pi_i = g(\pi_i) + r \quad \text{and} \quad g(\pi_i) > 0 \}.
\]

Fig. 3. Threshold for queueing for the Random Server algorithm and the Peeling algorithm with $p = 0.8$.

By solving the mean fields equations in Theorem 1, we obtain a table of $\sigma_p(r, f_i)$. With the values of $\sigma_p(r, f_i)$, we can obtain the queuing threshold $\rho^*$. For instance, with $p = 0.8$ and $q = 0.4$, $\rho^* = 0.695$ for the Random Server algorithm and $\rho^* = 0.765$ for the Peeling algorithm. Fig. 3 plots the thresholds for these two algorithms against $q$ with $p = 0.8$. It shows that with the same $p$ and $q$, the threshold under the Random Server algorithm is smaller than that under the Peeling algorithm. In addition, the thresholds for both algorithms increase to 1 as $q$ increases towards $p$, since the system behaves as a homogeneous system when $q$ almost equals $p$.

B. Effective mean service rate

We define the effective mean service rate $\mu_e$ as
\[
\mu_e = \frac{n}{m\pi_p + m\pi_q} = \frac{\lambda}{\pi_p + \pi_q}. \quad (12)
\]

\[
\mu_e = \frac{n}{m\pi_p + m\pi_q} = \frac{\lambda}{\pi_p + \pi_q}. \quad (12)
\]
which measures the efficiency of the servers. When the load is below the queueing threshold, $\mu_e$ can be obtained from Theorem 5. We do not have explicit characterization of the fixed points at high load. Instead, we iterate the mean-field equations over multiple time slots to obtain the fixed point.

Fig. 4 shows the effective mean service rate at the fixed points for both algorithms with $p = 0.6$ and $q = 0.2$. As soon as queueing takes place, the mean service rate for the Peeling algorithm decreases drastically and converges to that of the Random Server algorithm. This is due to the lack of degree-one server nodes and the peeling stage stops when there are still a large number of unassigned tasks.

VI. PERFORMANCE IN CONTINUOUS TIME MODEL

We evaluate the Peeling algorithm against the Random Server algorithm via simulation in continuous time. Consider a system of $m$ parallel servers. Tasks arrive at the system as a Poisson process with rate $\lambda$. The service time of a task assigned to a server with/without local data is assumed to i.i.d. with distribution $E_p$ ($E_q$) with mean $\frac{1}{\mu} (\frac{1}{\mu})$. Tasks are assigned in the following ways:

Random Server algorithm: When an arriving task sees some idle servers, an idle server is randomly selected for this task; otherwise this task joins the queue. When a server becomes idle, if its degree is zero, a task is sampled from the unassigned pool uniformly randomly; otherwise a task is selected randomly from the tasks that have data replicated on this server. If no unassigned task exists in the system, the server just stays idle.

Peeling algorithm: When a task arrives, if no idle servers are available, it joins the queue; otherwise a server with the least non-zero degree is selected to process this task. If all idle servers are of degree 0, this tasks is assigned to a randomly selected idle server. When a server becomes idle, it follows the same rule as the Random Server algorithm.

Consider exponential service time distribution. Fig. 5 shows the effective mean service rate. Similar to the results in Fig. 4, which is obtained from the computation using mean-field equations in the discrete-time model, the Peeling algorithm outperforms the Random Server algorithm at low to moderate load, while converges to the Random Server algorithm with load increasing. Note that these two algorithms show similar performance trend in different models. Hence performance analysis under the discrete-time model provides insight for the performance of these algorithms in real scenario, and also offers guidelines for the design of efficient algorithms.

VII. CONCLUSION

We proposed a degree-guided task assignment algorithm that is shown to significantly outperform the Random Server algorithm over the region of light to medium load. The future work is two-fold: (1) We plan to solve the fixed point problem at high load. (2) We are interested in designing an algorithm that outperforms the Random Server algorithm at high load.

Q. Xie is supported by AFRL grant FA8750-11-2-0084. Y. Lu is partially supported by NSF grant CNS-1150080.

ACKNOWLEDGMENT

Q. Xie is supported by AFRL grant FA8750-11-2-0084. Y. Lu is partially supported by NSF grant CNS-1150080.

REFERENCES