Models for Distributed, Large Scale Data Cleaning

Vincent J. Maccio, Fei Chiang, and Douglas G. Down

McMaster University
Hamilton, Ontario, Canada
{macciov,fchiang,downd}@mcmaster.ca

Abstract. Poor data quality is a serious and costly problem affecting organizations across all industries. Real data is often dirty, containing missing, erroneous, incomplete, and duplicate values. Declarative data cleaning techniques have been proposed to resolve some of these underlying errors by identifying the inconsistencies and proposing updates to the data. However, much of this work has focused on cleaning data in static environments. Given the Big Data era, modern applications are operating in dynamic data environments where large scale data may be frequently changing. For example, consider data in sensor environments where there is a frequent stream of data arrivals, or financial data of stock prices and trading volumes. Data cleaning in such dynamic environments requires understanding the properties of the incoming data streams, and configuration of system parameters to maximize performance and improved data quality. In this paper, we present a set of queueing models, and analyze the impact of various system parameters on the output quality of a data cleaning system and its performance. We assume random routing in our models, and consider a variety of system configurations that reflect potential data cleaning scenarios. We present experimental results showing that our models are able to closely predict expected system behaviour.

Keywords: data quality, distributed data cleaning, queueing models

1 Introduction

In the Big Data era, data quality has become a prolific issue spanning across all industries. Data-driven decision making requires having access to high quality data that is clean, consistent and accurate. Unfortunately, most real data is dirty. Inconsistencies arise due to the integration of data from multiple, heterogeneous data sources, where each source has its own data representation and format. Data inconsistencies also arise when integrity constraints, the rules defined over the data to keep the data accurate and consistent, are violated and not strictly enforced. When integrity constraints are not strongly enforced, there is no mechanism to validate whether the data updates are correct, thereby leading to data errors.

Existing data cleaning systems [1–7] have proposed techniques to repair the data by suggesting modifications to the data that conform to user expectations or to a given set of integrity constraints. However, most of these systems have focused on cleaning *static* instances of the data. That is, a snapshot of the data is taken, and if any changes in the data occur after the snapshot, then a new snapshot must be taken. While existing techniques may work well for static data environments, many modern applications are now operating in dynamic data environments where the data is frequently changing. For example, data in sensor, financial, retail transactions, and traffic environments require processing large scale data in near real-time settings. This has led to the need for more dynamic data cleaning solutions [8], particularly, those that can support large scale datasets.

In this paper, we present a set of models for distributed large scale data cleaning, where the data cleaning task is delegated over a set of servers. Such models are relevant for cleaning large data sets where the data can be partitioned and distributed in a parallel server environment. Each server has its own properties, such as the service time in which it cleanses the data, and the quality of the output data. We apply principles from queueing theory to model a variety of server configurations, and analyze the tradeoff between the system performance and data quality. We consider variations in our model, such as job priorities, and servers discarding data. Finally, we present our experimental results showing the accuracy of our model predictions against simulation results using real datasets.

This paper is organized as follows. In Section 2, we present related work, followed by preliminary background in Section 3. In Section 4, we present details of our distributed data cleaning model, and the variations we consider that reflect real application environments. Finally, we present our experimental results in Section 5, and conclude in Section 6.

2 Related Work

Recent data cleaning systems such as AJAX [2], Nadeef [3], LLUNATIC [9] and others, have focused on cleaning a static snapshot of the data for a given set of constraints. While these solutions are effective for data environments where the data and the constraints may not change frequently, they are expensive to implement in environments where the data and the constraints may evolve, as they require manual re-tuning of parameters, and acquisition of new data and constraints. In our work, we focus on modelling distributed data cleaning in dynamic environments, and study the influence of parameter changes on the data quality output.

As data intensive applications increasingly operate in data environments where rules and business policies may change, recent work in this area has proposed repair techniques to consider evolving constraints, primarily focused on functional dependencies (FDs) [10, 11]. However, the objective of these techniques is to propose specific modifications to the data and/or constraints to resolve the inconsistency. Given the increasing need for scalable data cleaning

systems, our objective is to analyze the behaviour of such a system under different parameter settings and configurations. To the best of our knowledge, none of the existing work has considered this.

3 Preliminaries

In this section, we provide a brief background on simple queueing models. A popular queueing model is the M/M/1 queue. This is used to represent a single server system with the following characteristics. First, jobs arrive to the system according to a Poisson process with rate λ, that is, the time between arriving jobs is exponentially distributed with rate λ. Second, jobs that arrive to the system are served one at a time following a First In First Out (FIFO) policy. Lastly, the time that it takes for a job to be processed by the server, often referred to as the *service time*, is exponentially distributed with rate μ. We refer to response time as the time from arrival to departure of a job in the system, and *service time* as the time to service (clean) a job once it enters the server. The two M's in the M/M/1 notation denote the characteristics of the arrival process and service time distribution, respectively, and represent for Markovian (or memoryless), while the 1 denotes that it is a single server system.

We can analyze this system as a Continuous Time Markov Chain, and expressions for the expected number of jobs in the system, $\mathbb{E}[N]$, as well as the expected response time, $\mathbb{E}[R]$, can be given by [12]:

$$\mathbb{E}[N] = \frac{\lambda}{\mu - \lambda} \quad \text{and} \quad \mathbb{E}[R] = \frac{1}{\mu - \lambda}. \tag{1}$$

The assumption of the exponential service times in the M/M/1 model can be limiting in some situations. In the M/G/1 queue, the service times follow a general distribution, G, where the first and second moments are known. Similar to the M/M/1 queue, closed form expressions for $\mathbb{E}[N]$ and $\mathbb{E}[R]$ are given by:

$$\mathbb{E}[N] = \rho + \frac{\rho^2 + \lambda^2 \sigma_S^2}{2(1-\rho)} \quad \text{and} \quad \mathbb{E}[R] = \frac{1}{\mu} + \frac{\rho^2 + \lambda^2 \sigma_S^2}{2\lambda(1-\rho)}, \tag{2}$$

where σ_S^2 denotes the variance of the service time distribution. More details about these and related models can be found in [12–14].

4 Distributed Data Cleaning

We present models for large scale data cleaning in dynamic data environments. We assume that there is some baseline data instance, and changes to the data are reflected by incremental updates to the baseline instance. These incremental updates occur with some frequency, which we call the *arrival rate*. As this data arrives, there is a data cleaning task, a *job*, that needs to be done using this data (in conjunction with the baseline data instance). Given a set of servers (each

with its own properties), and a set of data arrival streams, what is the optimal assignment of incoming data jobs to servers, such that the data quality output is maximized and the overall system performance is stable?

To solve this problem, we assume several random routing settings, where each server is independent. Furthermore, the decision of which server handles an incoming data job is made independently of the current state of the servers. If we do not make such assumptions, determining the optimal configuration would be intractable as it would be an extension of the "slow server problem" [15], a well-known problem in the queueing literature. We begin by considering a simple base model with two servers, and extend this model to consider other variations. For each variation, we provide analytic results expressing the performance vs. data quality tradeoff, and provide our insights on system behaviour. We also present our experimental results comparing our predicted model values against a simulation using a real energy metering data stream. We believe our work provides meaningful insights to a data analyst on the expected system behaviour and quality of the data cleaning task for large scale data.

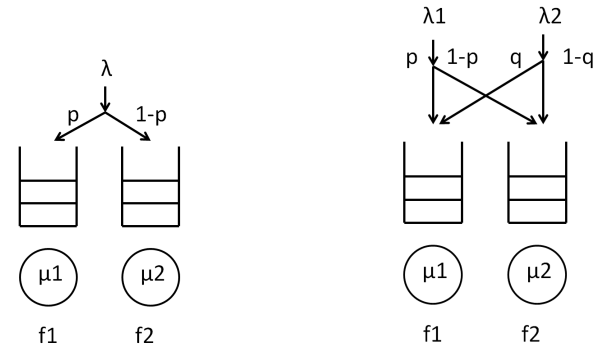
4.1 The Base Model

Consider a data cleaning system with two distinct servers, where jobs arrive according to a Poisson process with rate λ. When a job arrives, it is sent to the first server with probability p, and to the second server with probability $(1-p)$. From the demultiplexing of the arrival stream via routing, each server can be seen as an independent M/M/1 queue. Furthermore, each server has an associated failure probability, denoted f_1 and f_2, where $0 \leq f_1, f_2 \leq 1$. These values represent the probability that a server incorrectly cleaned the data. That is, after a job completes, the server either failed or the proposed updates to the data were incorrect. We define a random variable F, such that $0 \leq F \leq 1$, that denotes the proportion of jobs in the system that have been incorrectly cleaned in steady state, and the data remains dirty. Let $\mathbb{E}[F]$ denote the expected value of F. Figure 1(a) shows the base model.

We assume $\lambda < \mu_1 + \mu_2$ to ensure system stability, and that the user cannot control λ, μ_1, μ_2, f_1 nor f_2, but can set the value for the routing probability p. Since the user has control of p, the goal is to select a value for p that minimizes expected response time, and maximizes the expected data quality (by minimizing $\mathbb{E}[F]$). We define a cost function to analyze this tradeoff.

$$\mathcal{C}_{Base} = \mathbb{E}[R] + \beta\mathbb{E}[F] \tag{3}$$

where the value β represents the relative weight of the expected data quality over the expected response time. We assume β is given by the user. The cost $\mathcal{C}_{Base}$ models our objective to minimize system response time, and to minimize failure (poor data quality results). We consider $\mathcal{C}_{Base}$ as an initial cost function. As future work, we plan to extend $\mathcal{C}_{Base}$ to include more complex, constrained models that minimize $\mathbb{E}[R]$ subject to $\mathbb{E}[F]$ satisfying minimum threshold levels. We can determine closed form expressions for $\mathbb{E}[R]$ and $\mathbb{E}[F]$ by noting that the



(a) Two server routing

(b) Classes and priorities

Fig. 1. Graphical queueing models

inputs to the two queues are a demultiplexed Poisson process, and therefore can be viewed as M/M/1 queues. By applying equation (1) we obtain:

$$\mathbb{E}[R] = \frac{p}{\mu - p\lambda} + \frac{1-p}{\mu - (1-p)\lambda} \tag{4}$$

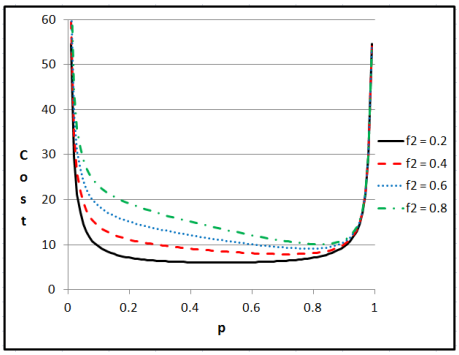
$$\mathbb{E}[F] = pf_1 + (1-p)f_2. \tag{5}$$

Equations (4) and (5) can be applied to compute expected response time and expected data quality output, respectively. Without loss of generality, such closed form expressions can also be computed for more complex, constrained models as mentioned above. Substituting (4) and (5) into (3) yields a closed form cost function.

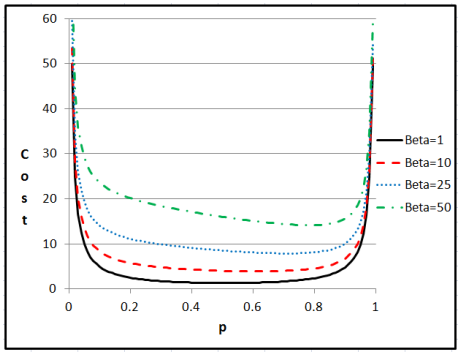
$$\mathcal{C}_{Base} = \beta(pf_1 + (1-p)f_2) + \frac{p}{\mu - p\lambda} + \frac{1-p}{\mu - (1-p)\lambda}. \tag{6}$$

Unfortunately, taking the derivative of (6), and setting it to zero does not yield a closed form expression in terms of p. However, we can analyze the tradeoff between p and the remaining parameters in (6) by plotting the cost $\mathcal{C}_{Base}$ against p for several configurations, as shown in Figure 2. We will refer to the optimal value of p as p^*.

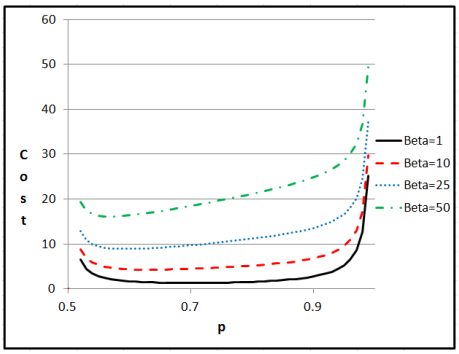
Figures 2(a) and 2(b) show two system configurations where each system has data jobs arriving at the same rate in which they can be processed ($\lambda = \mu_1 = \mu_2$). In Figure 2(a), the second server produces lower quality data than the first server ($f_1 \leq f_2$). In Figure 2(b), there is a greater weight (β) on improved data quality than system response time. In both Figure 2(a) and 2(b), p should be selected at the point where the cost is minimal. We observe that in both graphs, the



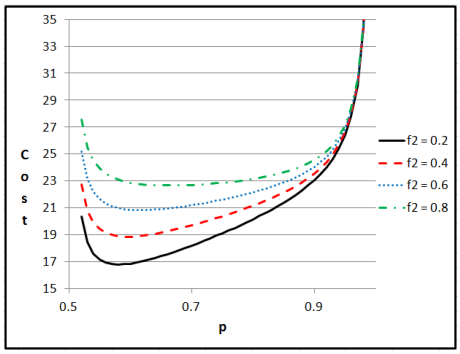
(a) $\mu_1 = 2, \mu_2 = 2, \lambda = 2, \beta = 25, f_1 = 0.2$



(b) $\mu_1 = 2, \mu_2 = 2, \lambda = 2, f_1 = 0.2, f_2 = 0.4$



(c) $\mu_1 = 4, \mu_2 = 2, \lambda = 4, f_1 = 0.5, f_2 = 0$



(d) $\mu_1 = 4, \mu_2 = 2, \lambda = 4, \beta = 25, f_1 = 0.9$

Fig. 2. Base model configurations

cost is fairly low for a wide range of p values, ranging from [0.1, 0.9], indicating that both systems are fairly stable. It is only at the endpoints ($p \to 0, p \to 1$) where the cost sharply increases and the system becomes unstable. Hence, a conservative approach is to assign arriving data jobs equally to both servers ($p = 0.5$) to maintain system stability.

Figure 2(c) and Figure 2(d) show less stable systems (than Figures 2(a) and 2(b)), where selecting a different p value yields more dramatic effects on the cost, and ultimately in the system response times. Selecting p must be done carefully to minimize the cost, as can be seen in Figure 2(d), where for $p > 0.6$, steeper curves reflect increasing instability in the system.

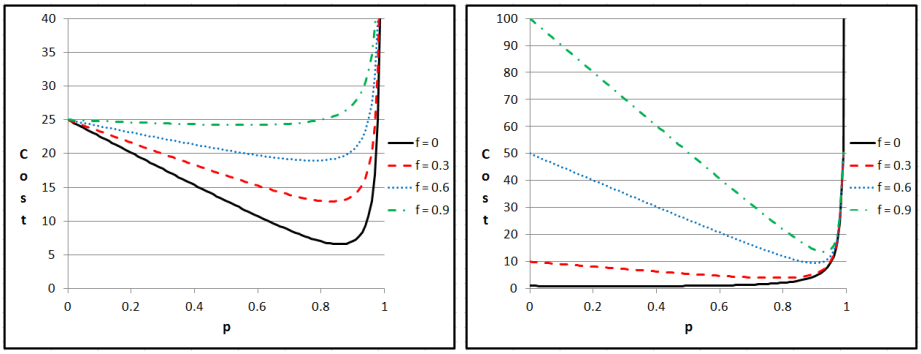
4.2 The Discard Model

In this section, we consider the case where a server may choose to *discard* particular data jobs. We consider discarding jobs for two reasons:

1. If the overall system is overloaded ($\lambda > \mu_1$), then arriving data jobs must be discarded to resume stability.
2. When service level agreements (SLAs) must be met, some incoming data jobs may need to be discarded.

At the routing step, if the decision is made to discard a job, then the failure rate $f = 1$, and the response time of the discarded job is instantaneous. There is no longer a restriction on λ to ensure stability, as incoming jobs can simply be removed from the system until the server is able to handle the incoming data. The discard model can be viewed as an instantiation of the base model (described in Section 4.1), where $f_2 = 1$ and $\mu_2 \to \infty$. The resulting cost function is,

$$\mathcal{C}_{Discard} = \beta(p(f_1 - 1) + 1) + \frac{p}{\mu - p\lambda} \tag{7}$$



(a) $\mu = 2, \lambda = 2, \beta = 25$ (b) $\mu = 2, \lambda = 2, f = 0$

Fig. 3. Discard model configurations

Figure 3(a) shows a system at full capacity ($\mu = \lambda$). The value f is varied from 0 (where the system always produces clean data and jobs are not discarded) to 0.9 (where most of the data will be incorrectly cleaned). We observe that for $f = 0$, with only one server, the system is quite unstable as seen by the wide fluctuation in cost values for $p \in [0.1, 0.9]$. As $f \to 0.9$, the system performance improves as incoming data jobs may be discarded to achieve stability.

In Figure 3(b), we investigate the influence of β on the overall system performance (recall β is a weight of the relative importance between data quality vs. system response time). As β ranges from [1,100], we want the system to produce increasingly higher quality output. For increasing β values, the system becomes increasingly unstable, as shown by the increasingly steep (negative sloped) curves. This indicates that in such a single server system, if we want to achieve improved data quality, we must be willing to tolerate wide fluctuations in system response time and stability.

4.3 Classes and Priorities

The models we have considered thus far include a single data arrival stream, and follow a FIFO policy. In this section, we explore how adding another data arrival stream, and including priorities influence the system behaviour. We consider two incoming data streams, where jobs still follow a FIFO policy. The data jobs arrive to each server at a rate of λ_1 and λ_2, and both are assumed to be Poisson processes. Each of the servers service incoming jobs following an exponential distribution, but depending on which stream the job arrived from, it may process the jobs with different service rates. We consider this preferential notion to model *priority* between data jobs, which exists in many applications systems to guarantee service level agreements.

For simplicity, we assume that the two servers are homogeneous with respect to their processing rates. Therefore, μ_1 and μ_2, denote the service rate of data jobs arriving from the first and second stream, respectively. We let p and q represent the probability that a data job is sent to the first server from the first and second stream, respectively. Similarly, an incoming job is sent to the second server with probability $(1-p)$ and $(1-q)$, from the first and second streams, respectively. Hence, a data analyst must determine appropriate values for p and q to maximize the data quality output from the servers, and minimize system response time. The system can be seen in Figure 1(b).

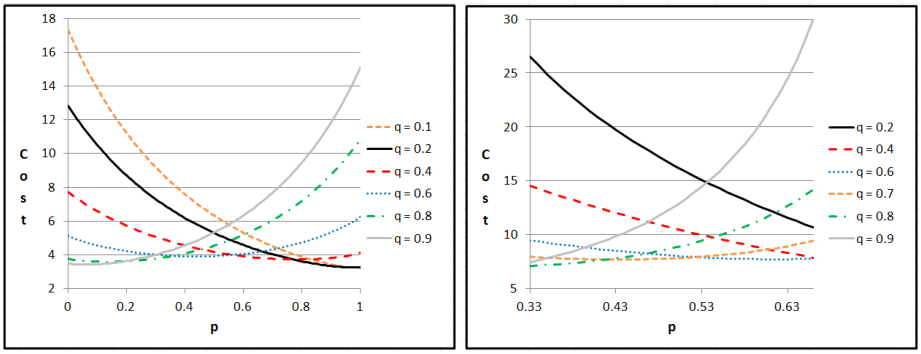
Our analysis here differs from prior models as we no longer have two M/M/1 queues, due to the differing service rates μ_1 and μ_2. However, we observe that while the service time distribution is not exponential, it is hyper-exponential. Therefore, an M/G/1 queue can be fitted to the system, and the Pollaczek-Khinchine formula, (2), can be used. This leads to the following cost function,

$$\begin{aligned}\mathcal{C}_{Classes} = \beta((p+q)f_1 + (2-p-q)f_2) + (p+q)\left(\frac{\rho_1}{\lambda_1^*} + \frac{\rho_1^2 + (\lambda_1^*)^2\sigma_{S1}^2}{2\lambda_1^*(1-\rho_1)}\right) \\ + (2-p-q)\left(\frac{\rho_2}{\lambda_2^*} + \frac{\rho_2^2 + (\lambda_2^*)^2\sigma_{S2}^2}{2\lambda_2^*(1-\rho_2)}\right),\end{aligned}$$

where,

$$\begin{aligned}\lambda_1^* &= p\lambda_1 + q\lambda_2 \\ \lambda_2^* &= (1-p)\lambda_1 + (1-q)\lambda_2 \\ \rho_1 &= \frac{\lambda_1^*}{\frac{p\lambda_1}{p\lambda_1+q\lambda_2}\mu_1 + \frac{q\lambda_2}{p\lambda_1+q\lambda_2}\mu_2} \\ \rho_2 &= \frac{\lambda_1^*}{\frac{(1-p)\lambda_1}{(1-p)\lambda_1+(1-q)\lambda_2}\mu_1 + \frac{(1-q)\lambda_2}{(1-p)\lambda_1+(1-q)\lambda_2}\mu_2} \\ \sigma_{S1}^2 &= \left(\frac{p}{\mu_1} + \frac{q}{\mu_2}\right)^2 + 2pq\left(\frac{1}{\mu_1} - \frac{1}{\mu_2}\right)^2 \\ \sigma_{S2}^2 &= \left(\frac{1-p}{\mu_1} + \frac{1-q}{\mu_2}\right)^2 + 2(1-p)(1-q)\left(\frac{1}{\mu_1} - \frac{1}{\mu_2}\right)^2.\end{aligned}$$

The physical interpretation of these parameters are: λ_1^* and λ_2^* are the arrival rates, ρ_1 and ρ_2 are the utilizations, and σ_{S1}^2 and σ_{S2}^2 are the variances of the service time distributions, for the first and second server, respectively. These follow from equation (2), and the definitions of the hyper-exponential distribution [12].



(a) $\mu_1 = 6, \mu_2 = 1.5, \lambda_1 = 2, \lambda_2 = 1 f_1 = 0.05, f_2 = 0.2, \beta = 25$

(b) $\mu_1 = 4, \mu_2 = 1.25, \lambda_1 = 2, \lambda_2 = 1 f_1 = 0, f_2 = 0.5, \beta = 5$

Fig. 4. Modelling with priorities

Using the above formulas, we plot a set of configurations as shown in Figure 4. In both Figures 4(a) and 4(b), the optimal parameter setting has a small value for q (approximately 0.2), but a large value for p (approximately 0.9). In both configurations, the first server is more reliable and produces higher quality data ($f_1 < f_2$). Our recommendation in this case is to assign incoming jobs to the first server without overloading it, so as to minimize the cost and maintain system stability. The remaining jobs can be sent to the second server so that it does not remain idle.

We now consider priorities between the incoming data streams, also referred to as *classes*. Although the analysis becomes more difficult, it remains tractable. Consider the system in Figure 1(b), where jobs that arrive from the first stream have higher priority than (i.e., they preempt) jobs from the second stream. Furthermore, if a high priority job arrives to the system while a low priority job is being processed, it preempts its execution, and takes its place.

We assume two priority classes: low and high. Each arriving high priority job views the system as a simple M/M/1 queue where only high priority jobs exist, with arrival rate $p\lambda_1$. We restrict the two job classes to share the same service time distribution, relative to the server. That is, μ_1 represents the service rate, for jobs of both classes, at the first server, while μ_2 represents the service rate for jobs of both classes at the second server. Since traditional M/M/1 queues

can now be applied, we view the arrival rate as $p\lambda_1 + q\lambda_2$. Interestingly, we can derive expressions for the number of jobs in the system, for each priority class, as given below.

$$\mathbb{E}[N] = \mathbb{E}[N_{high}] + \mathbb{E}[N_{low}]$$

where $\mathbb{E}[N]$ denotes the number of jobs in a specific queue. Due to the previous observation that high priority jobs can be modelled as an M/M/1 queue with arrival rate $p\lambda_1$, we can derive the number of low priority jobs at the first queue.

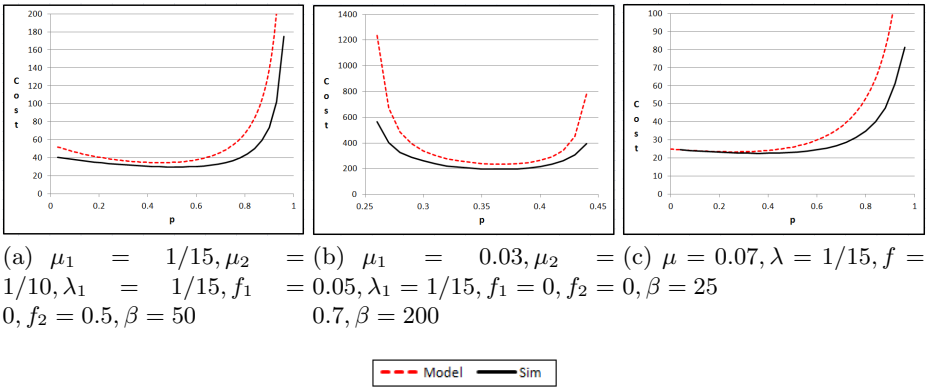
$$\mathbb{E}[N_{low}] = \frac{p\lambda_1 + q\lambda_2}{\mu_1 - p\lambda_1 - q\lambda_2} - \frac{p\lambda_1}{\mu_1 - p\lambda_1}. \tag{8}$$

Equation 8 allows us to compute the expected number of low priority jobs in the system. In addition to the expression for $\mathbb{E}[N]$, we can specify constraints on the system that limit the total number of (low priority) jobs. For example, if we have SLAs, we can impose constraints on the system that allow a proportion of the servers to only handle low priority jobs. Furthermore, given the arrival rate, service rate, and probabilities p and q, we can directly compute the estimated number of jobs per class, which can enable a data analyst to predict system behaviour. In addition, we can also consider how the server failure rates, per job class priority, will be affected by changes in these parameters. For example, high priority jobs should be serviced by reliable servers where f is close to 0. We plan to investigate this direction as future work.

5 Experiments

In this section, we validate the accuracy of our proposed models by comparing the optimal routing probabilities (p^*), to the values given by our simulated system. In our simulations, we assume our data arrival streams follow a Poisson process. We simulated 100,000 arriving jobs using the CSIM 19 library [16]. Our simulation was run on a server with Intel Xeon E5-2960 processors, 4 vCPUs, and 16GB of RAM. We used real data describing energy usage across our University campus, that reported values such as water, hydro usage, and air temperature readings. The readings are reported every 15 minutes.

Figure 5 shows our simulation results. Figure 5(a) shows a lightly loaded system where the base model is used. We observe that the shape of the curves are quite similar, and as expected, the cost from the simulation is less than that of the model. The value of p^*, predicted by our model is within 10% of the simulated p value. If we chose to use the predicted routing probability from the queueing model, our results in the simulated system would yield a cost that is within 10% of the minimum. We note that the difference in p^* values occurs in the insensitive portion of the curve ($p \in [0.05, 0.4]$) where the cost is relatively constant, and the model predictions closely follow the simulation results. Our results in Figure 5(b) show very promising results. Figure 5(b) shows a heavily



(a) $\mu_1 = 1/15, \mu_2 = 1/10, \lambda_1 = 1/15, f_1 = 0, f_2 = 0.5, \beta = 50$ (b) $\mu_1 = 0.03, \mu_2 = 0.05, \lambda_1 = 1/15, f_1 = 0, f_2 = 0, \beta = 25$ (c) $\mu = 0.07, \lambda = 1/15, f = 0.7, \beta = 200$

Fig. 5. Experiments

loaded base model system, which can become unstable depending on the choice of p. We found that the values of p^* predicted by the queueing and simulation models were equal.

In our last experiment, we simulated the discard model. Figure 5(c) shows that the values of p^* from the queueing and simulations models differ by approximately 0.08. If we apply the values predicted by the queueing model in our system, our cost would be within 2.5% of the minimal cost for this workload. Overall, our evaluation has provided very promising initial results showing that our models are able to closely predict optimal routing probabilities for incoming data cleaning tasks. By having these optimal values, data analysts are able to better understand and predict anticipated system load, and stability, in order to maximize the data quality output from the system.

6 Conclusion

In this paper, we have taken a first step towards modelling a distributed data cleaning environment where each job represents the incoming data that needs to be cleaned with respect to a baseline data instance. We have presented a set of models that reflect different data cleaning system configurations. Specifically, we have presented analytic models, expressions, and insights for a base model, discard model (where jobs can be discarded), and a priority-class model. We have investigated the stability (and response time) to data quality tradeoff for each of these models, and revealed some interesting insights. For example, we have observed that particular configurations can lead to unstable system performance, and provided cases when discarding incoming jobs may be necessary. Our evaluation revealed promising accuracy results, where our model predictions are all within 10% of the simulated results. Avenues for future work include further model validation, extending the models to consider n servers, and investigating

constrained models where the parameters are subject to a set of cost or threshold constraints.

References

1. Raman, V., Hellerstein, J.M.: Potter's wheel: An interactive data cleaning system. In: VLBD. (2001) 381–390
2. Galhardas, H., Florescu, D., Shasha, D., Simon, E., Saita, C.A.: Declarative data cleaning: Language, model, and algorithms. In: VLDB. (2001) 371–380
3. Dallachiesa, M., Ebaid, A., Eldawy, A., Elmagarmid, A., Ilyas, I.F., Ouzzani, M., Tang, N.: NADEEF: A commodity data cleaning system. In: SIGMOD. (2013) 541–552
4. Bohannon, P., Fan, W., Flaster, M., Rastogi, R.: A cost-based model and effective heuristic for repairing constraints by value modification. In: SIGMOD. (2005) 143–154
5. Yakout, M., Elmagarmid, A.K., Neville, J., Ouzzani, M., Ilyas, I.F.: Guided data repair. VLDB Endowment **4**(5) (2011) 279–289
6. Chiang, F., Miller, R.J.: A unified model for data and constraint repair. In: ICDE. (2011) 446–457
7. Chiang, F., Wang, Y.: Repairing integrity rules for improved data quality. IJIQ (2014) 20 pages
8. Volkovs, M., Chiang, F., Szlichta, J., Miller, R.J.: Continuous data cleaning. In: ICDE. (2014) 12 pages
9. Geerts, F., Mecca, G., Papotti, P., Santoro, D.: The LLUNATIC data-cleaning framework. PVLDB **6**(9) (2013) 625–636
10. Chiang, F., Miller, R.J.: Active repair of data quality rules. In: ICIQ. (2011) 174–188
11. Beskales, G., Ilyas, I.F., Golab, L., Galiullin, A.: On the relative trust between inconsistent data and inaccurate constraints. In: ICDE. (2013) 541–552
12. Gross, D., Harris, C.M.: Fundamentals of Queueing Theory. Third edn. Wiley-Interscience (1998)
13. Harchol-Balter, M.: Performance Modeling and Design of Computer Systems: Queueing Theory in Action. Cambridge University Press (2013)
14. Kleinrock, L.: Queueing Systems. Volume 1. Wiley-Interscience (1975)
15. Rubinovitch, M.: The slow server problem. Journal of Applied Probability **22**(4) (1985) 205–213
16. Mesquite Software CSIM 19. http://www.mesquite.com/