Preprint published in the *Future Generation Computer Systems*. DOI: 10.1016/j.future.2018.06.033

Toward a Smart Data Transfer Node

Zhengchun Liu[a,*], Rajkumar Kettimuthu[a], Ian Foster[a,b], Peter H. Beckman[a]

[a] *Mathematics & Computer Science Division, Argonne National Laboratory, Lemont, IL 60439, USA*
[b] *Department of Computer Science, University of Chicago, Chicago, IL 60637, USA*

Abstract

Scientific computing systems are becoming significantly more complex, with distributed teams and complex workflows spanning resources from telescopes and light sources to fast networks and Internet of Things sensor systems. In such settings, no single, centralized administrative team and software stack can coordinate and manage all resources used by a single application. Indeed, we have reached a critical limit in manageability using current human-in-the-loop techniques. We therefore argue that resources must begin to respond automatically, adapting and tuning their behavior in response to observed properties of scientific workflows. Over time, machine learning methods can be used to identify effective strategies for autonomic, goal-driven management behaviors that can be applied end-to-end across the scientific computing landscape. Using the data transfer nodes that are widely deployed in modern research networks as an example, we explore the architecture, methods, and algorithms needed for a smart data transfer node to support future scientific computing systems that self-tune and self-manage.

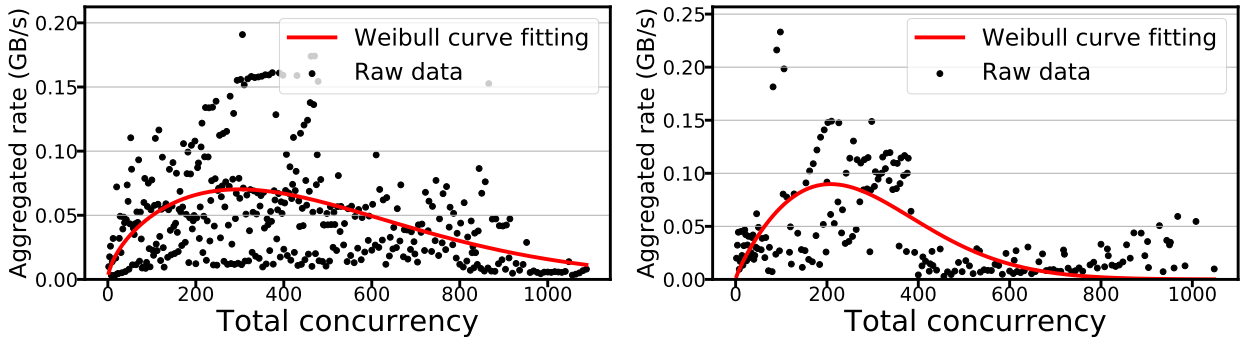
Keywords: Data transfer node, Smart computing, File transfer, Reinforcement learning

1. Introduction

Scientific computing systems are becoming significantly more complex and have reached a critical limit in manageability using current human-in-the-loop techniques. To address this situation, we need to devise autonomic, goal-driven management actions, based on machine learning, applied end-to-end across the scientific computing landscape. The high-performance computing center was previously the nexus of the scientific computing universe, both administratively and computationally. Users brought their codes and their data to computing facilities, and the operational teams managing the systems carefully configured and monitored systems to achieve the required uptimes and queue wait times. As science workflows get complex, spanning distributed resources and involving a distributed team of researchers, no single, centralized administrative team and software stack can coordinate and manage all the resources. Thus, smart systems that achieve self-configuration, self-optimization, self-healing, and self-protection have garnered the attention of researchers in both academia and industry [1, 2, 3, 4, 5].

Data transfer nodes (DTNs) [6] are compute systems dedicated for data transfers in distributed science environments. In previous work [7, 8], we determined via the analysis of millions of Globus [9] data transfers involving thousands of DTNs that DTN performance has a nonlinear relationship with load. Aggregate DTN throughput first increases with transfer load but, after a threshold, decreases because of overload (see Figure 1). A DTN thus has an optimal operating point. As instantaneous DTN load is determined by the characteristics of the transfers that are currently running, which are in turn determined by the parameters of those transfers, the optimal operating point cannot easily be determined analytically.

* Corresponding author: Zhengchun Liu, Bldg. 240, Argonne National Lab., 9700 S. Cass Avenue, Lemont, IL 60439, USA
Email addresses: zhengchun.liu@anl.gov (Zhengchun Liu), kettimut@anl.gov (Rajkumar Kettimuthu), foster@anl.gov (Ian Foster), beckman@mcs.anl.gov (Peter H. Beckman)



(a) DTN at NERSC

(b) DTN at JLab

Figure 1: Aggregate incoming transfer rate vs. total concurrency (i.e., instantaneous number of GridFTP server instances) at two heavily used endpoints, with Weibull curve [10] fitted. Figure source: [7].

An ideal scheduling algorithm will ensure that a given DTN always works in its optimal operating point. since resources used to transfer data over wide area network—for example, networks and storage systems at the source and destination endpoints are shared with other applications—a static policy is powerless to deal with these dynamics. Analytical methods are inadequate in capturing the collective impact of application parameters because these parameters are often categorical and their impact on performance and power usage is opaque to such methods [11]. We report here on a preliminary study in which we apply a deep reinforcement machine learning–based knowledge engine to power a DTN with smartness in order to achieve self-awareness, self-configuration, and self-optimization. Our goal is to make the DTN always run at its optimal operating point (or at least avoid overloading) if there are sufficient transfer tasks. The key difference between this paper and other studies [12, 13] on optimizing wide-area data transfer performance is that we try to maximize the aggregated throughput of a DTN, whereas others try to optimize the throughput of a given transfer.

The rest of the paper is organized as follows. In §2 we present the architecture of a smart cyberinfrastructure in which each subsystem has the ability to act autonomously. In §3 we detail the design and implementation of a smart DTN. Our experiment results are discussed in §4, where we present two experiments to show the effectiveness of the knowledge engine. In §5 we review related work, and in §6 we summarize our conclusions and briefly discuss future work.

2. Motivation

Extraordinary advances in computing, communication networks, and information technologies have produced an explosive growth of highly interconnected systems, which are increasingly becoming complex, dynamic, heterogeneous, labor intensive, and challenging to operate and manage with existing approaches [14]. For large organizations such as DOE's science community, with thousands of geographically interconnected systems, traditional distributed systems operation and management based on static behaviors, interactions, and configuration are proving to be inadequate. We define a system architecture in which, as shown in Figure 2, each edge resource has its own knowledge engine (KE). This component acts as the "brain" of an edge resource, generating control commands based on the current system state as determined from monitoring data and on learned knowledge of the relationship between actions and cost/benefits. Thus, the KE has three key components: (1) input features that reflect the current state of the physical system, (2) output control commands that steer the physical system to operate optimally for current tasks, and (3) machine learning models that capture and optimize the relationship between input and output.

Figure 2 shows the an architecture of science autonomous infrastructure, which consists of five layers: fabric, perception, processing, collective, and application. The **fabric layer** consists of the resources or things in the smart science ecosystem: for example, computational resources, storage systems, network

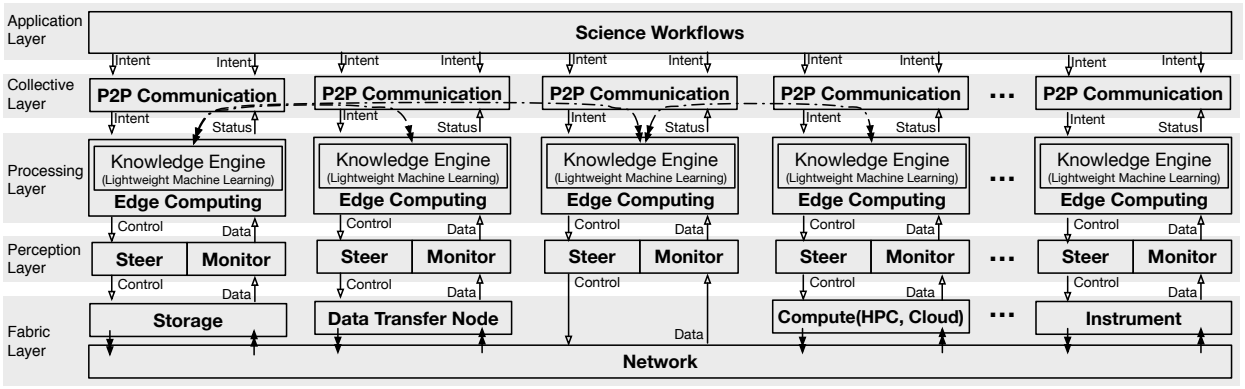


Figure 2: Smart HPCC architecture.

resources, and instruments. The **perception layer** gathers static and dynamic information about the resources to discover the state, structure and capabilities as well as provides mechanism to steer (or perform actions on) the resources. The *processing layer* stores, analyzes, and processes data that comes from the perception layer. It can use learning algorithms to detect patterns, find anomalies, predict performance and make proactive decisions. The **collective layer** defines the necessary communication and authentication protocols and mechanisms for the exchange of data among the KEs in the processing layer of different resources in the science ecosystem. The **application layer** comprises of the user applications within the smart science ecosystem.

As one can see, each component in Figure 2 is powered by a KE capable of sensing information from the environment and/or other components and optimizing itself by learning from its history. In the current era of distributed and data-intensive science, data movement is a critical aspect. Thus, in this work we focus on designing a smart data transfer node, a first step toward designing a smart distributed science ecosystem.

Here, a DTN is typically a PC-based Linux server, built with high-quality components and configured specifically for wide-area data transfer. It is a key component in distributed science environments [15], and its performance has direct influence on the productivity of the whole ecosystem. A DTN typically mounts the parallel file system that serves the compute cluster and is connected to a high-speed wide area network. For a given transfer, a DTN either pulls data from the storage and sends the data over its network interface card or receives data from its network card and writes the data to its storage system. Heavily used DTNs, such as those at national supercomputer centers, serve as both source and destination for multiple concurrent transfers [16]. Based on our extensive study of millions of Globus transfer logs [7, 17], we find a big space for improvement in data transfer performance by optimizing DTN behavior. Self-configuration and self-optimization are two of the key features we achieve in this paper. Basically, these features mean that the DTN is self-aware and knows how to steer itself. The following are a few examples of ways in which a smart DTN can adapt its behavior to optimize desired outcomes.

- **Network packet pacing.** Packet pacing can improve performance [18], but its value depends on the characteristics of the destination endpoint. Typically, it is desirable only when transferring data from a higher bandwidth endpoint to a lower bandwidth endpoint. Since a DTN may transfer data to endpoints of different types (e.g., other DTNs, desktops, laptops), a smart DTN should apply pacing differentially to different edges (source to destination endpoint pair) based on destination capabilities.

- **File transfer order.** Overall performance can be improved by rearranging the order in which files are transferred based on the file layout in an object-based storage system such as Lustre and/or by reducing the long tail with concurrent transfers.

- **Edge processing.** A smart DTN may perform computation to optimize transfer performance. For example, if it detects that network connectivity has become a bottleneck, it may interact with a DTN at the other side to evaluate the effectiveness of transferring compressed data.

- **Network congestion control.** Currently, no versatile TCP congestion control algorithm exists that performs optimally for any kind of network condition. Other protocols, such as UDT [19], perform better in a lossy network environment. A smart DTN will learn from its history to select the best algorithm or protocol for each edge.

- **Self-configuration.** Liu et al. [7] observe that each DTN has an optimal operating point (see Figure 1. This optimal point varies not only across DTNs but also with transfer characteristics. A smart DTN will try to tune itself to work at the optimal point.

- **Data transfer parameters.** A transfer tool may have multiple tunable parameters whose value not only affects transfer performance but also decides DTN occupancy. The overall DTN performance, for example, total bytes transferred per day, is highly related with its load. Improper scheduling of tasks makes the DTN overloaded and operate inefficiently [7].

In this preliminary work, we consider the last item, which directly determines DTN state and load, to make a DTN continuously operate in its optimal point.

3. Smart data transfer node

The smartness of a DTN is achieved by using its self-awareness ability. Self-awareness includes actively sensing the current state of the environment and discovering knowledge about the cost and benefits of its configurations. Figure 3 demonstrates the work process of a smart DTN. The KE uses a "chunk," or portion of a file, as a scheduling unit instead of the entire file. The KE will vary the size of each chunk dynamically depending on the current state of the system. It will adaptively determine the number of concurrent chunks to be transferred at any instance. Moreover, it will determine the tunable parameters to be used for each chunk.

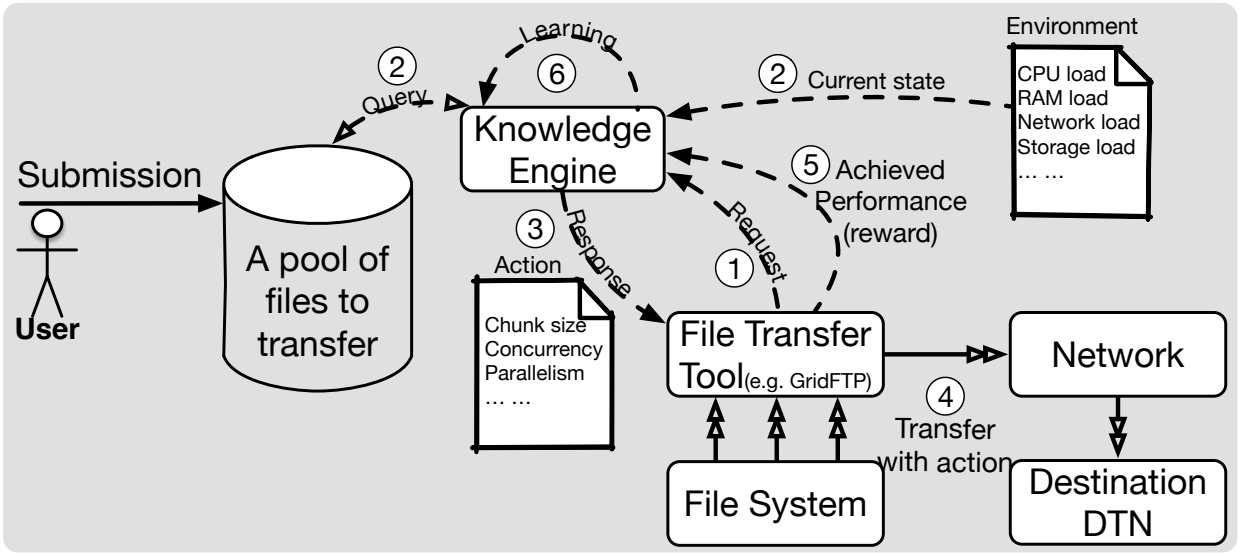


Figure 3: Work process of a smart DTN. The architecture of the KE and step 6 are detailed in Figure 4.

As the "brain" of a smart DTN, the KE maintains information about the current state and knowledge about data transfer behavior. As shown in Figure 3, we explain the work process of a smart DTN and its self-learning process as follows. ① A file transfer tool requests a file to transfer from the KE. The KE ② checks the current DTN state (detailed in §3.2) and ③ responds to the transfer tool with a chunk of file and corresponding optimal transfer parameters (the steering action). ④ The transfer tool transfers the

associated chunk with the parameters and monitors the aggregate DTN throughput during this transfer. ⑤ Once completed, DTN's average aggregate throughput is reported to the KE as a reward for its actions. ⑥ Based on the reward (encourage or discourage), the KE updates its internal model parameters to improve its decision policy (details are shown in Figure 4).

3.1. Reinforcement learning

Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward [20]. It is an area of machine learning inspired by behaviorist psychology, concerned with how software agents ought to act in an environment so as to maximize some notion of cumulative reward. Reinforcement learning is different from supervised learning that learns from examples provided by a knowledgeable external supervisor. In the reinforcement learning model, learning occurs from interaction. It is often impractical to obtain examples of desired behavior that are both correct and representative of all situations. The actor will eventually be in uncharted territory and must learn from its own experience.

Google DeepMind developed a solid algorithm to tackle the continuous action space problem. Building on the prior work of [21] on deterministic policy gradients, they produced a policy-gradient actor-critic algorithm called Deep Deterministic Policy Gradients [22] that is off-policy (does not need to follow any specific policy) and model-free (does not learn the underlying dynamics that govern how an agent interacts with the environment). Specifically they use the algorithm to represent the policy function independently of the reward function. The policy function structure is known as the actor (μ), and the reward function is referred to as the critic (Q). The actor produces an action based on current state, and the critic produces a temporal-difference error signal given the state and resultant reward. If the critic is estimating the action-value function, it will also need the output of the actor. The output of the critic drives learning in both the actor and the critic. Neural networks are used to represent the actor and critic structures in deep reinforcement learning. Therefore, state, actions, and rewards are three key considerations when we use reinforcement learning to solve a problem. Basically, action is taken to get maximum reward based on current state. In the following sections we explain DTN's state, actions for steering data transfer, and reward to evaluate an action.

3.2. DTN state

A DTN's state includes G, the total number of files being transferred concurrently (in our case, the number of GridFTP instances); S, the total number of TCP streams; K_{in} and K_{out}, the aggregate ingress and egress throughput of the DTN's network interface card that quantify the DTN's network load, respectively; and D_r and D_w, the aggregate disk read and write throughput, respectively. K_{in} and K_{out} are determined by monitoring via *ifconfig*. D_r and D_w are determined by reading the */proc/[pid]/io* of all processes.

3.3. Actions

One action is comprised with three variables (like three tunable knobs), with raw values as follows in this paper: (1) whether to transfer the chunk at all, {`true`, `false`}; (2) if the first variable is `true`, the chunk size to transfer, in the range [200M, 1000M]; and (3) the parallelism for the transfer, in the range [1, 8]. For use in the neural network, each raw value is scaled to the range 0 to 1, with, in the first case, a value ≥ 0.5 representing true and < 0.5 representing false. The first variable indirectly controls the total concurrency on the DTN, as if `true`, a new GridFTP process is started to transfer the chunk, incrementing the total concurrency by one. If the first variable is `false`, then the system waits for a period (in the studies reported here, 10 seconds) and then returns to ask the KE again.

We note that the optimal parameters are determined based on the state before the transfer start. However, the DTN load (e.g., a transfer completed) and environment (e.g., storage, network condition) change continuously. Thus, the optimal parameters will no longer be optimal when the state changes. Since the current implementation of GridFTP does not support dynamically changing transfer parameters, we use the chunk-by-chunk scheme to mimic the necessary approach. Arguably, this scheme causes considerable overhead, because there is a GridFTP startup cost (about 2 seconds on our testbed) for transferring each chunk.

3.4. Reward

An action is the set of parameters assigned to a transfer. Our goal is to maximize DTN performance. We define the reward assigned to a given action as the average aggregated throughput observed during the associated transfer. Specifically we mention two cases. (1) If the first action variable is `true` and it ends with taking T seconds to transfer bytes given by the second action variable, the reward will be the average aggregated throughput during this T seconds. (2) Otherwise, if the first action variable is `false`, which means that the current concurrency is already an optimal configuration, the reward will be the average aggregated throughput during the following 10 seconds.

3.5. Learn and act process

A reinforcement learning model is trained by encouraging good actions and discouraging bad actions, evaluated by reward. Figure 4 illustrates the learning and action process.

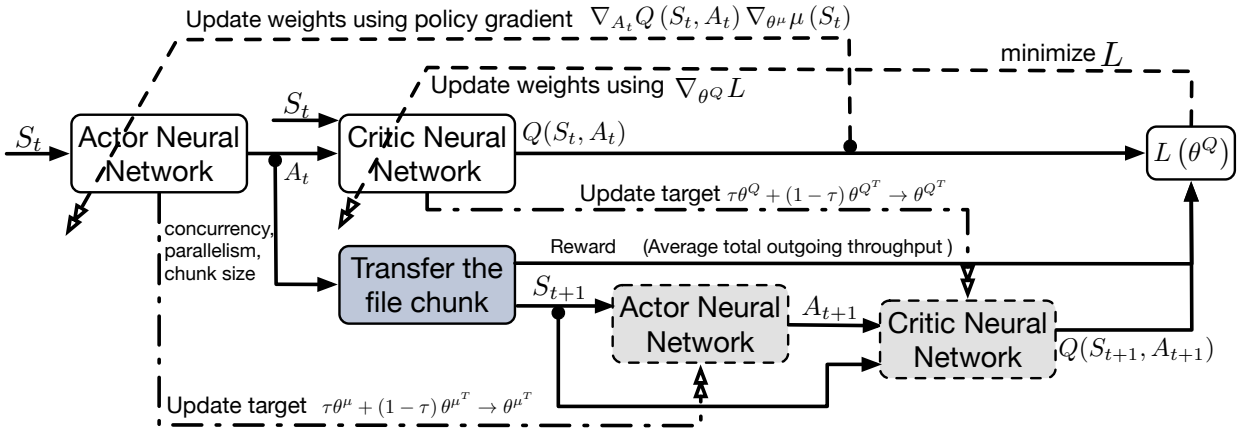


Figure 4: Using deep deterministic policy gradients [21] on a DTN to dynamically optimize tunable parameters. Dashed boxes correspond to target neural networks. In this paper we use Globus GridFTP to transfer the file chunk. We note that, term t represents the transfer for tth file chunk.

Specifically, we train the critic neural network by minimizing the cost function via the Bellman equation [21, 22]:

$$L\left(\theta^Q\right) = \frac{1}{N} \sum_t \left[Q(S_t, A_t) - y_t\right]^2 \tag{1}$$

where $Q(S_t, A_t)$ is the value predicted by the critic neural network right after the actor makes the decision A_t (i.e., the action as explained in §3.3) for transfer t and where y_t is the Bellman function:

$$y_t = r\left(S_t, A_t\right) + \gamma Q(S_{t+1}, A_{t+1}), \tag{2}$$

where r denotes the actual reward we got after the transfer t by using parameters in A_t; γ represents the discount factor; and $Q(S_{t+1}, A_{t+1})$ is the future reward predicted by critic after the transfer t at state S_{t+1}. Basically, we update the weights of the actor network in order to make it generate actions that accord with the Bellman equation.

We update the actor neural network by using the sampled policy gradient (computed by using the chain rule to the expected return with respect to the actor weights), which was proved to be the gradient of the policy's performance [21].

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_t \nabla_{A_t} Q\left(S_t, A_t\right) \nabla_{\theta^\mu} \mu\left(S_t\right) \tag{3}$$

We update the weights of the target actor neural network μ^T (Equation 4) and the target critic neural network Q^T (Equation 5) by having them slowly track the learned neural network [22]. Here we use $\tau = 0.9$ for both networks.

$$\theta^{\mu^T} \leftarrow \tau\theta^{\mu} + (1-\tau)\,\theta^{\mu^T} \tag{4}$$

$$\theta^{Q^T} \leftarrow \tau\theta^{Q} + (1-\tau)\,\theta^{Q^T} \tag{5}$$

3.6. Our reinforcement learning model

Our actor neural network comprises three fully connected hidden layers with 400, 300, and 200 neurons, respectively, with a rectified linear unit (ReLU) activation function and with a sigmoid activation function on the output layer. For the critic, since it takes both the state and the action as inputs, we use three hidden layers with 400, 400, and 300 neurons, respectively, to connect to the state input. The action input connects to two hidden layers, of sizes 400 and 300, respectively. Their outputs are added and then connect to the output layer. All neurons in the hidden layers have ReLU activation. There is no activation function (i.e., linear activation) for the output neuron because it is a regressor. We implement the neural networks with Google Tensorflow [23], with all weights initialized randomly, and we use the Adam optimizer [24] with learning rate 0.0001 for both the actor and critic.

In reinforcement learning, the learner is not told which actions to take; instead, the learner must discover the actions that yield the most reward by trying them [20]. In order to avoid getting stuck in a local minimum, there is a tradeoff between exploration (of uncharted territory) and exploitation (of current knowledge). The exploration helps the learner explore new areas of the space of possible actions, but it may give bad rewards. The rewards are guaranteed in exploitation, but there are probably better actions. In this work, we give a 5% chance for exploration; in other words, 5% of the actions are randomly assigned. Furthermore, since samples generated from exploring an environment sequentially are not independently and identically distributed, we use a replay buffer—a finite-sized cache that saves the most recent chunk transfers [22]. In each training epoch, training samples are sampled uniformly from the replay buffer.

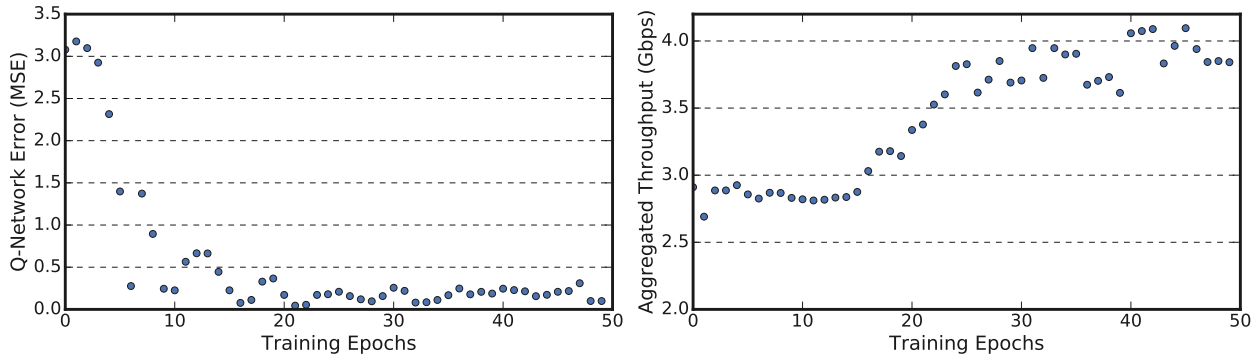
4. Experimental results

We first conduct experiments over two dedicated DTNs. Figure 5a shows the convergence of the critic neural network. Since our batch size is 64 (i.e., we retrain the neural network each time there are 64 new samples), the KE is able to generate optimal actions after training with 1,280 file-chunk transfers (20 epochs). Since the knowledge engine accurately improves itself while the environment keeps changing continuously, our replay buffer size has been set as 1,000; in other words, the knowledge engine should forget very old information.

4.1. Dedicated environment

To permit controlled and reproducible experiments, we work in an environment in which the source DTN, network, and destination DTN are not shared with any other activity.

To test the effectiveness of our KE, we run our first experiment in an isolated environment where only our testing data transfer service is running (i.e., no other program competes for resources with the data transfer service). In this case, more optimal operating means better transfer performance. Figure 5 shows the effectiveness of KE. We see that overall transfer performance, which is the reward to train the reinforcement learning model, increases as the KE becomes more accurate. We note that Figure 5b is the average throughput between two epochs; that is, some transfers finished during the ith epoch are initiated during previous epoch when the KE is not as accurate as the ith epoch. Thus the benefit in Figure 5b we observed comes later than the improvement of KE accuracy in Figure 5a.



(a) Q-network prediction error (MSE)

(b) Average aggregated throughput (reward)

Figure 5: Effectiveness of the KE in a dedicated environment. DTN performance increases as the KE's prediction accuracy improves.

4.2. Shared environment

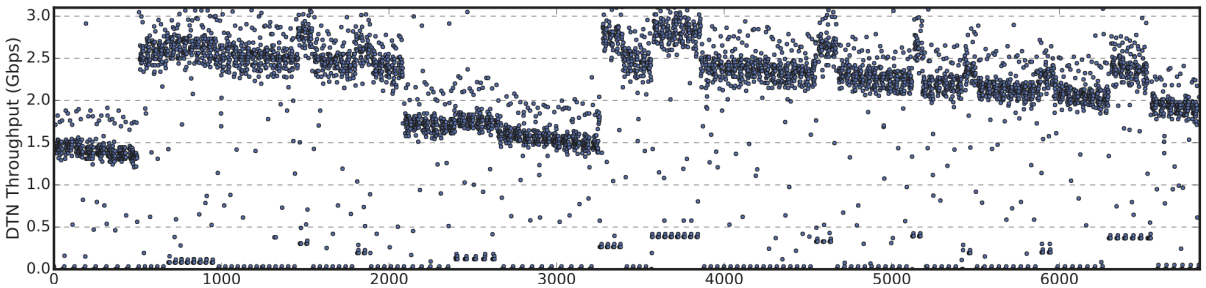
Most DTNs operate in a shared environment, meaning that they are subject to a mix of controlled load (i.e., transfers for which the DTN can control transfer parameters) and other load (i.e., transfers and file system accesses that occur periodically, but over which the DTN has no control. To permit controlled experimentation in such environments, we configure our testbed to permit the application of reproducible external load, that is, load that is unknown to our KE but is generated via a random number generator with fixed random seed so that it can be replayed as required. The dataset to transfer in our experiments consists of 150 files totaling 1.5 TB.

Figure 6a and Figure 6b compare aggregate DTN throughput when subject to a mix of known load and unknown external load, with and without the KE. We see that the *mean* performance stays roughly the same in the two cases but that the variance is reduced when using the KE. What exactly is going on? We believe that the KE is identifying parameters that improve performance, particularly when aggregate transfer rates are lower, but that the impact of these improvements is being counteracted, in terms of the contributions to overall mean performance, by the increased overheads that result, in our implementation, from restarting GridFTP for each chunk in order to change parameters. In Figure 6a, there are only 150 GridFTP startup costs for the 150 files. In Figure 6b, we split the 150 files into 5,404 chunks to enable selection of optimal parameter values for each chunk. (Recall that the current GridFTP implementation does not support changing transfer parameters dynamically.)

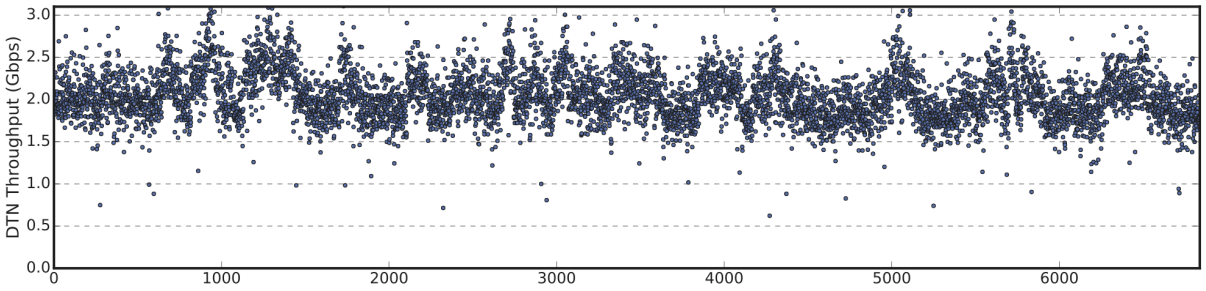
If our GridFTP implementation could reconfigure transfer parameters dynamically, without reestablishing data channels (TCP streams) each time, these overheads could be removed and the overall average throughput improved. To simulate the likely impact of such optimizations, we adjusted the transfer rates used to compute the aggregate throughput in Figure 6b by using steady-state throughput to replace start-up throughput during the start-up in the logs. The results, in Figure 6c, suggest that we can get at least 11.3% improvement compared with the heuristic approach in Figure 6a. Figure 7 shows the cumulative distribution of throughput achieved with these three approaches, in which the aggregated throughput is sampled every second. As one can see, with heuristic approach, there is about 20% of chance to see the aggregated throughput less than 300 Mbps. However, with KE and the overhead caused by tool implementation, the chance to see the aggregated throughput less than 1.5 Gbps is less than 1%.

5. Related work

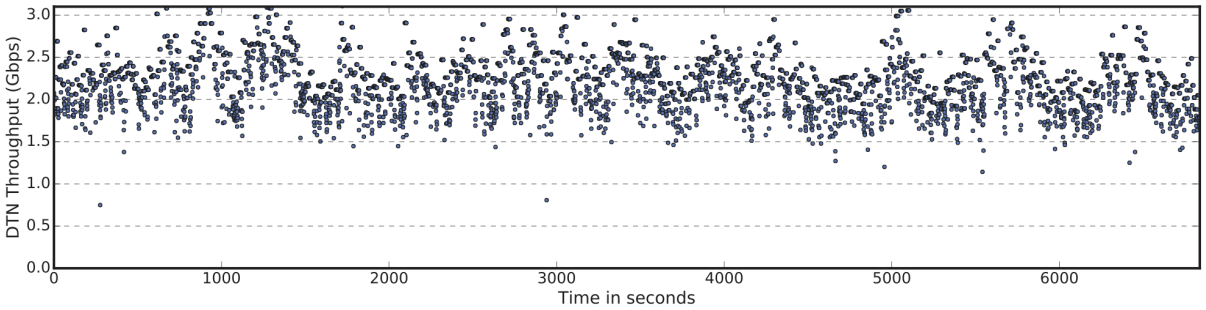
IBM pointed out in 2001 [2] that the difficulty of managing today's computing systems goes well beyond the administration of individual software environments. They suggested the concept of *autonomic computing* [1] to describe computing systems that are said to be self-managing, and they argued for self-configuration, self-optimization, self-healing, and self-protecting as four key properties.



(a) Case when transfer parameter values are fixed. Overall average is 2.040 Gbps.



(b) Case when transfer parameter values are set dynamically with the KE. Overall average is 2.043 Gbps.



(c) Results from (b), adjusted to remove chunk overheads. Overall average is 2.273 Gbps. (Some points overlap.)

Figure 6: Transfer performance over a roughly two-hour period in three scenarios: (a) transfer parameters are fixed; (b) transfer parameters are set dynamically with the KE; and (c) the data from (b) adjusted to remove file chunk overheads. Each point represents per-second aggregate DTN throughput. Each figure shows results with the same external workload (emulated).

Parashar et al. [25] studied autonomic Grid applications that are context aware and are capable of self-configuring, self-composing, self-optimizing, and self-adapting. Huebscher et al. [4] surveyed concentrations of research emerging in key applications areas. Salehie et al. [26] reviewed fundamentals of autonomic computing and discussed three different types of complexity that address characteristics of autonomous systems mentioned in literature.

CometCloud [3] is an autonomic framework designed to enable software-defined federations for dynamic and data-driven end-to-end workflows. Its key layers are infrastructure/federation, autonomic management, and programming/interface. The autonomic management layer lets users and applications define objectives and policies that drive resource provisioning and application workflow execution, while satisfying user constraints.

Recent years have seen a rapid growth of interest in exploiting monitoring data and applying machine learning for automated management and performance analysis [27]. Kettimuthu et al. [5] reviewed recent work that uses machine learning algorithms to improve computer system performance, and identified the gaps and open issues. They proposed a hierarchical architecture that builds on the earlier proposals for autonomic computing systems to realize an autonomous science infrastructure.

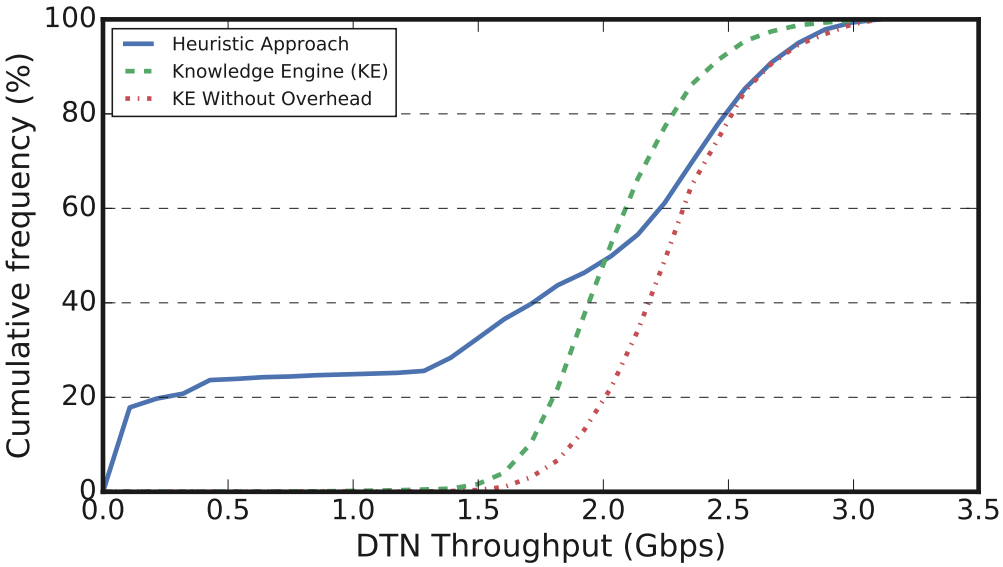


Figure 7: Comparison of throughput achieved with heuristics approach, KE, and KE without overhead by using the cumulative distribution function.

As for large data transfer over wide-area network, many publications examine methods to optimize the performance of an individual transfer by tuning application-level parameters. Liu et al. [28] developed a tool to optimize multifile transfers by opening multiple GridFTP threads. Their tool increases the number of concurrent flows up to the point where the transfer performance degrades. Their work also proved that the number of concurrent transfer files can only benefit a transfer to some extent, it causes negative influence after that optimal point. However, their work focuses only on concurrent file transfers; other transfer parameters such as number of parallel streams per file are not considered.

Kim et al. [29] proposed an application-layer throughput optimization model based on prediction of parallel TCP streams. It relies on real-time network probing, which either causes too much sampling overhead or fails to accurately predict the correct transfer parameters for long-elapsed transfers because of dynamic network condition. Engin et al. [30] combined historical data analysis with real-time sampling that enables their algorithms to tune the application level data transfer parameters accurately and efficiently to achieve close-to-optimal end-to-end data transfer throughput with low overhead.

Engin et al. [13] clustered files by size and then used a heuristic approach to estimate the optimal Globus application-level parameter values (i.e., pipelining, parallelism, and concurrency) to be used in each cluster, in order to maximize the overall transfer throughput in wide area networks. Specifically, based on file characteristics and real-time investigation, their algorithms dynamically tune parallelism per file, the level of control channel pipelining, and the number of concurrent file transfers to increase I/O throughput. Nine et al. [12] used historical knowledge about the network and data to reduce the real-time investigation overhead while ensuring near optimal throughput for each transfer. They mined historical transfer logs to perform knowledge discovery about the transfer characteristics. Then, when about to start a new transfer, they used the discovered knowledge from the offline analysis along with real-time investigation of the network condition to optimize the protocol parameters. However, this work relies on real-time investigation, which is expensive to be used to optimize the aggregated throughput of a DTN because a given DTN may contact with many other DTNs simultaneously.

Clearly, all these works focus on optimizing the performance of a given transfer. However, the total throughput of a given DTN is limited by hardware, and mostly there are multiple simultaneous transfers to and from the DTN , overlapped with the transfer of interest. Increasing the throughput of one transfer via tuning its transfer parameters may decrease the throughput of other transfers. At the end, the aggregated throughput of the DTN will be affected, making the DTN operate at a non-optimal point. This paper is

motivated from a perspective of overall performance. The goal is to make DTN always operate at its optimal point.

Reinforcement learning has achieved great progress in the past few years. For example, AlphaGo used reinforcement learning [31] to beat the world No. 1 ranked player at the time in a three-game match. Lillicrap et al. [22] adapted the ideas underlying the success of deep Q-Learning to the continuous action domain. They presented an actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces. Their algorithm is able to find policies whose performance is competitive with those found by a planning algorithm with full access to the dynamics of the domain and its derivatives. Kim et al. [32] applied reinforcement learning to autonomous helicopter flight. They proved that reinforcement learning can deal with this challenging control problem with high-dimensional, complex, asymmetric, noisy, nonlinear, and dynamics difficulties.

6. Conclusion

We have proposed a smart data transfer node that uses deep reinforcement learning to learn the relationship between transfer parameters and overall throughput. We argue that such a system can identify transfer parameter values that achieve higher overall performance than simple heuristics can. Our results suggest that a knowledge engine that implements such methods can indeed guide a data transfer node to stable, sustained transfer performance.

While these results are encouraging, the work reported here is preliminary and has many limitations. For example, we consider only a single experimental configuration and do not compare against more complex heuristics. Our autonomic control methods manage only data transfer parameters—but, as discussed in §2, many other actions (e.g., network configuration strategies, such as packet pacing and congestion control; edge-processing methods, such as data compression before transfer) remain to be explored in order to steer more optimally. We hope to address these and other limitations in future work.

Acknowledgment

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357 and the DOE AMASE project fund by SmartHPCC program managed by Thomas Ndousse-Fetter. We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

References

[1] J. O. Kephart, D. M. Chess, The vision of autonomic computing, Computer 36 (1) (2003) 41–50.
[2] P. Horn, Autonomic computing: IBM's perspective on the state of information technology.
[3] J. Diaz-Montes, M. AbdelBaky, M. Zou, M. Parashar, CometCloud: Enabling software-defined federations for end-to-end application workflows, IEEE Internet Computing 19 (1) (2015) 69–73.
[4] M. C. Huebscher, J. A. McCann, A survey of autonomic computing–degrees, models, and applications, ACM Computing Surveys 40 (3) (2008) 7.
[5] R. Kettimuthu, Z. Liu, I. Foster, P. H. Beckman, A. Sim, J. Wu, W. keng Liao, Q. Kang, A. Agrawal, A. Choudhary, Toward autonomic science infrastructure: Architecture, limitations, and open issues, in: the 1st Autonomous Infrastructure for Science workshop, AI-science 2018, ACM, New York, NY, USA, 2018. doi:10.1145/3217197.3217205.
URL http://doi.acm.org/10.1145/3217197.3217205
[6] https://www.es.net, Science DMZ: Data Transfer Nodes, https://fasterdata.es.net/science-dmz/DTN/.
[7] Z. Liu, P. Balaprakash, R. Kettimuthu, I. Foster, Explaining wide area data transfer performance, in: 26th ACM Symposium on High-Performance Parallel and Distributed Computing, 2017. doi:10.1145/3078597.3078605.
[8] Z. Liu, R. Kettimuthu, I. Foster, N. S. Rao, Cross-geography scientific data transfer trends and user behavior patterns, in: 27th ACM Symposium on High-Performance Parallel and Distributed Computing, HPDC '18, ACM, New York, NY, USA, 2018. doi:10.1145/3208040.3208053.
URL http://doi.acm.org/10.1145/3208040.3208053
[9] K. Chard, S. Tuecke, I. Foster, Efficient and secure transfer, synchronization, and sharing of big data, IEEE Cloud Computing 1 (3) (2014) 46–55.
[10] W. Weibull, A statistical distribution function of wide applicability, Journal of Applied Mechanics (1951) 293–297.

[11] A. Marathe, R. Anirudh, N. Jain, A. Bhatele, J. Thiagarajan, B. Kailkhura, J.-S. Yeom, B. Rountree, T. Gamblin, Performance modeling under resource constraints using deep transfer learning, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '17, ACM, New York, NY, USA, 2017, pp. 31:1–31:12. doi:10.1145/3126908.3126969.
URL http://doi.acm.org/10.1145/3126908.3126969

[12] M. S. Zulkar Nine, K. Guner, Z. Huang, X. Wang, J. Xu, T. Kosar, Data transfer optimization based on offline knowledge discovery and adaptive real-time sampling, ArXiv e-printsarXiv:1707.09455.

[13] E. Arslan, T. Kosar, A heuristic approach to protocol tuning for high performance data transfers, ArXiv e-printsarXiv:1708.05425.

[14] Z. Liu, R. Kettimuthu, S. Leyffer, P. Palkar, I. Foster, A mathematical programming- and simulation-based framework to evaluate cyberinfrastructure design choices, in: IEEE 13th International Conference on e-Science, 2017, pp. 148–157. doi:10.1109/eScience.2017.27.
URL http://doi.org/10.1109/eScience.2017.27

[15] E. Dart, L. Rotman, B. Tierney, M. Hester, J. Zurawski, The Science DMZ: A network design pattern for data-intensive science, Scientific Programming 22 (2) (2014) 173–185.

[16] R. Kettimuthu, Z. Liu, I. Foster, Y. Liu, A comprehensive study of wide area data movement at a scientific computing facility, in: 2018 Scalable Network Traffic Analytics, SNTA'18, IEEE, 2018.

[17] R. Kettimuthu, Z. Liu, D. Wheelerd, I. Foster, K. Heitmann, F. Cappello, Transferring a petabyte in a day, Future Generation Computer Systems (2018) 10.

[18] B. Tierney, N. Hanford, Recent Linux TCP updates, and how to tune your 100G host, in: SC16 INDIS Workshop.
URL https://scinet.supercomputing.org/workshop/sites/default/files/100G-Tuning-INDIS.tierney.pdf

[19] Y. Gu, R. L. Grossman, UDT: UDP-based data transfer for high-speed wide area networks, Computer Networks 51 (7) (2007) 1777–1799.

[20] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, Vol. 1, MIT press Cambridge, 1998.

[21] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, Deterministic policy gradient algorithms, in: Proceedings of the 31st International Conference on Machine Learning (ICML-14), 2014, pp. 387–395.

[22] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, CoRR abs/1509.02971.
URL http://arxiv.org/abs/1509.02971

[23] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous distributed systems, CoRR abs/1603.04467.
URL http://arxiv.org/abs/1603.04467

[24] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, CoRR abs/1412.6980.
URL http://arxiv.org/abs/1412.6980

[25] M. Agarwal, V. Bhat, H. Liu, V. Matossian, V. Putty, C. Schmidt, G. Zhang, L. Zhen, M. Parashar, B. Khargharia, et al., Automate: Enabling autonomic applications on the grid, in: Autonomic Computing Workshop, IEEE, 2003, pp. 48–57.

[26] M. Salehie, L. Tahvildari, Autonomic computing: Emerging trends and open problems, SIGSOFT Softw. Eng. Notes 30 (4) (2005) 1–7. doi:10.1145/1082983.1083082.
URL http://doi.acm.org/10.1145/1082983.1083082

[27] G. Casale, Accelerating performance inference over closed systems by asymptotic methods, Proc. ACM Meas. Anal. Comput. Syst. 1 (1). doi:10.1145/3084445.
URL http://doi.acm.org/10.1145/3084445

[28] W. Liu, B. Tieman, R. Kettimuthu, I. Foster, A data transfer framework for large-scale science experiments, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, ACM, New York, NY, USA, 2010, pp. 717–724. doi:10.1145/1851476.1851582.
URL http://doi.acm.org/10.1145/1851476.1851582

[29] J. Kim, E. Yildirim, T. Kosar, A highly-accurate and low-overhead prediction model for transfer throughput optimization, Cluster Computing 18 (1) (2015) 41–59.

[30] E. Arslan, K. Guner, T. Kosar, HARP: predictive transfer optimization based on historical analysis and real-time probing, in: SC'16, 2016, pp. 25:1–25:12.
URL http://dl.acm.org/citation.cfm?id=3014904.3014938

[31] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of Go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489.

[32] H. J. Kim, M. I. Jordan, S. Sastry, A. Y. Ng, Autonomous helicopter flight via reinforcement learning, in: Advances in neural information processing systems, 2004, pp. 799–806.