Project Summary

A growing and inevitable trend in the use and management of storage resources is the consolidation of distributed storage into a large data center, which provides shared data access service. This practice is in response to the rapidly growing cost for labor-intensive system administration as well as for high resource utilization. Despite its compelling advantages and years of efforts on research and product development, storage consolidation has not yet been widely accepted in today's computing. One of the critical challenges to turn this promising technology into reality is that the quality of the I/O service (QoS) cannot be conveniently and efficiently guaranteed for users who outsource their dedicated storages to the shared system. To this end, a performance interface, or specification of required service quality, must be able to (1) allow a diverse set of users to easily relate each of their I/O QoS to their respective application performance; (2) enable efficient system implementation to meet the QoS requirements.

Currently the QoS requirements are usually presented in the form of service-level agreement (SLA) to bound latency and throughput of I/O requests. While SLA can be suitable for service-level requests to application servers, it is not for I/O requests from application servers to storage servers for two major reasons. First, users have difficulties to determine appropriate latency and throughput bounds to ensure their application performance. Overshooting the requirements can lead to excessive charge and unjustified system load. In addition, these bounds are inherently related to data access pattern, which users can hardly predict. Second, for the storage system, resources allocation can be misguided by the SLA measures because they do not consistently reflect users' resource demands. This makes resource provisioning and scheduling less informative and greatly reduces system efficiency.

To address these technical challenges, in this CAREER proposal the PI proposes *reference storage device* as performance interface and will implement consolidated storage service based on the interface. When a user creates a virtual storage device hosted in a shared system, its performance is associated with a reference storage device, which can be a user's originally dedicated system, instead of a set of (artificial) bounds. A user is then guaranteed to receive an I/O service whose quality is at least as good as the reference system regardless of variations and instantaneous changes of data access patterns.

The **intellectual challenges** of the project for building virtual storage devices of efficient QoS guarantee in a consolidated storage system are fourfold: (1) The automated abstraction of performance characteristics of a reference system must be efficient and of acceptable accuracy, which defines the QoS goals for the consolidated system to meet. (2) Reducing interference among virtual devices is a fundamental and difficult issue. Scheduling algorithms with high efficiency are needed to minimize the overhead for a high system utilization. (3) Modern storage systems are complex and hybrid, consisting of hard disks, solid-state disks, and other types of devices. Mapping a virtual device of specific QoS requirements to a physical array of matching performance characteristics for high resource utilization and strong QoS assurance demands intelligent algorithms to track and analyze resource usage. (4) To make the proposed reference storage system truly effective for data centers, a prototype with new algorithms will be developed and tested in large distributed systems.

If the project is successful, its **broader impact** will be significant: (1) The proposed research will remove the significant barrier for users and system administrators to communicate I/O performance requirements and pave the road to wide acceptance of the storage consolidation technology. (2) Gaining the insights into interactions among performance requirements of virtual devices and resource provisioning of the shared system will not only lay out an important foundation for implementing efficient virtual devices, but also provide valuable guidance for administration of any storage systems. (3) The research training to both undergraduate and graduate students, especially to under-represented minority students in the Southeast Michigan area, will address the concerns of lacking strong data and storage system professionals in the IT industry. (4)A major component in the CAREER proposal is to integrate the proposed research into PI's education activity. Our intensive curriculum development will not only address several limitations in existing computer system teaching, but also introduce related research results to classrooms in a timely fashion.

1 Introduction

"QoS specifications for storage systems appear to need to be rather richer than their counterparts in the networking space, probably because of the much greater potential for non-linear performance interactions on mechanical storage devices, and caches."

—— From a report on a decade-long project on automated management of enterprise storage systems at HP Labs [100]

1.1 Background and Problem Statements

Current technology trend in the IT industry shows that computing becomes more and more data intensive [4, 26, 46, 86]. However, it can be prohibitive for each individual users or divisions within an organization to own and maintain their separate storage systems. Besides hardware expense and power expense, system administration can cost several times the purchase price of the storage hardware [12, 16]. In addition, a storage system dedicated to a small group can lead to low utilization. By consolidating the distributed storage into a data center where I/O services are provided to users as utility, users are relieved from the complexity of management work, such as backup, archival, snapshot, and data migration, and benefit from better economies of scale. Seeing that storage consolidation provides a promising solution to the rising cost in today's data-centric computing, the IT industry quickly responds by providing all types of architectures of consolidated storages, from the file-level NAS [40] to the block-level SAN [10], from local enterprise storage servers to Amazon's S3 on-line storage service [2]. Today's technology is ready to create data centers comprising of tens of large arrays, connected by high-speed, switched fabrics, multi-petabytes of capacities, and potential aggregate bandwidth of tens of GB/s.

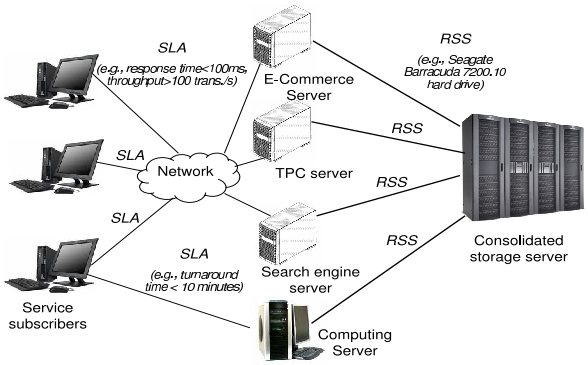


Figure 1: There are three tiers in a typical system architecture involving consolidated storage service: service subscribers, application servers, and storage server. While the SLA-like performance interface describing bounds on response time and throughput is suitable between service subscribers and application servers, a different performance interface is needed between application servers and storage server, such as the reference storage system (RSS) proposed in this project.

While the technology is ready on the architectural and operational aspect, there is a critical barrier to keeping performance-sensitive users from using the service, which is the lack of an effective performance interface through which users can present their goals on quality of I/O service (QoS) and the system can efficiently meet the goals. As an example, Amazon's cloud computing service (Amazon EC2) [1] allows users to specify a virtual computing system (named as *instance* in EC2) to be hosted in Amazon's virtual computing environment and upload their applications to run in the virtual system. All major aspects of an instance affecting applications' performance can be clearly specified *except* I/O performance. For example, in a small instance there are 1.7 GB memory, one processor equivalent to a 2007 Intel Xeon, and 160 GB

storage. Since storage system are shared among instances, the I/O performance can only be specified with an indicator, either *moderate* or *high*. This vague QoS specification will deter many users from using the service to run their I/O-intensive applications, since it leaves applications' performance highly unpredictable.

Figure 1 illustrates a typical service system architecture, which contains three tiers: service subscribers, application servers, and consolidated storage server. The consolidated storage server can provide I/O service either at the block level such as SAN or at the file-level such as NAS. We focus on the block-level storage, as it provides potential benefit to all applications that use either file systems or raw block interfaces to access data. Service subscribers request application-level services from application servers, such as browsing/ordering transaction requests to an E-Commerce server, or execution of a program in a computing server. Between these two tiers, the performance interface is usually in the form of service-level agreement (SLA), which allows users to specify their QoS requirements and for systems to measure delivered service quality in terms of bounds on response time and throughput of requests. While service subscribers can easily correlate their desired performance to theses measures and thus determines specific bounds, the SLA interface serves as a good bridge between these two tiers.

By treating application servers issuing I/O requests as the clients and consolidated storage server as the server, the SLA specification in terms of response time or throughput bounds is currently also used as a performance interface between these two tiers. However, this raises a serious issue in the QoS management of the two tiers. As service-level performance of applications running on the application servers, such as throughput of transactions or execution time of a scientific program, can be easily measured to see whether the SLA performance goals from service subscribers are met, it is very hard, if not impossible, to determine corresponding response time or throughput bounds for the I/O requests issued to the storage server so that the SLA goals are met. This is because many aspects of the I/O service are so dynamic that any static bounds cannot capture the real performance demands on individual requests.

First, the I/O requests can be very bursty. Applications may be designed to aggregate its I/O operations into several dedicated I/O phases, such as phases for data loading or data writing back, in anticipation that I/O system would be more efficient by serving requests in batch. The response time of a request issued in a busy period tends to be much higher than that in a quiet period. This is usually not a performance issue with users, who would also have the experience with their dedicated system. However, if a static response time bound is specified, all requests are treated indiscriminately on the same performance target. It would be unrealistic for the server to keep response time from growing with the bursty requests considering limited resource has to be used economically. If the target is determined based on requests issued in the quiet period, which actually imposes artificially strict latency bound on bursty requests, the resource has to be lavishly provisioned to meet the bound and the I/O service would be prohibitive to users. In other words, the users would receive performance benefit (a latency that does not go up with the request rate) they do not ask for at an unaffordable cost. A makeshift solution is to specify a set of performance bounds, each associated with a request rate. This puts the burden on the users to quantify the relationship between request rate and expected response time, as well as find out how this is translated into the service-level performance, such as program's execution time. These are all very challenging tasks for users.

Second, the request size can be highly variable. Requests of different sizes, or, the amount of requested data, should come with different respond time bounds. As the request service time can substantially rely on request size [1] [80], one fixed bound for all requests is fundamentally inaccurate. Determination of separate bounds for requests of sizes in different ranges is a tedious work and not flexible. A bandwidth bound, which specifies amount of data accessed in the unit time (MB/s) instead of number of I/O operations in the unit time (IOPS) per second is not subject to variation of request size. However, if a storage system guarantees the bandwidth bound, application designers would not have the incentive to aggregate small requests into one large request and compromise storage system's efficiency.

Last but not least, spatial locality of requests can be varying. Spatial locality describes the sequentiality of continuously requested data on the storage device. Usually this concept applies to the hard-disk-based

[1] It is noted that the service time is not proportional to request size for the disk-based storage system because disk seek time is independent of request size.

storage. The hard disk is such a device that access of sequential data is at least one order of magnitude faster than access of random data, because significant seek time is usually involved in accessing non-sequential data. The spatial locality is a highly elusive property that is very hard for users to characterize.

If the bounds are determined by always assuming a random access pattern, users have to be conservative in setting the bounds, and the server would also conservatively plan its capability for the worst scenario. If the actual access pattern turns out to be sequential access, the application is likely to only receive performance as low as that with random access when the server resource is tight with high I/O demand. This is against the users' experience with their dedicated storage systems, that is, I/O performance increases as spatial locality of accessed data improves. One consequence of this experience is that users are not motivated to take efforts in optimizing their programs by replacing small and random I/O access with large and sequential access, which definitely causes fatal impact on the efficiency of the storage server.

Consider the scenario where the bounds are determined by always assuming a sequential access pattern but the actual access is random. As it can take more than ten times disk service time for the same request when it is random than when it is sequential, the actual demand on the server's capability increases over ten times to meet the same pre-specified static bounds. If the server plans its provisioning based on the bounds with sequential access and there are a number of applications actually sending random requests, the server can quickly be overloaded and probably miss the performance bounds, especially for applications who do send sequential requests, breaking the principle of performance isolation. The idea to require users determine multiple performance bounds, each for one different spatial locality, is not practical. First, it is very hard for users to quantify the relationship between I/O performance and spatial locality, though the conceptual relationship is clear. Second, spatial localities can be mixed [36]. A limited set of bounds cannot capture the variations.

In summary, while the functional interface of consolidated storage to its clients has been well developed [13], the performance interface is still in its infancy. Current consolidated storage service either is limited to the sharing of storage space with best-effort service quality, or simply adopts the SLA performance interface. The performance interface plays the role of the pricing mechanism in a market, which should be expressive for consumers to evaluate cost effectiveness of the service they request and correctly reflect the producers' cost to guide the market to run in the most efficient manner. Furthermore, the performance interface as a pricing mechanism should encourage clients to optimize their I/O operations and enable efficient implementation of quality of service. If service charge is based on performance bounds that is independent of access pattern, the relationship between service consumers and producers is distorted, which renders the entire system inefficient. This has severely prevented users from widely adopting the technology in spite of compelling advantages.

1.2 Executive Summary of Proposed Research

The PI proposes an innovative project to address the critical issues and challenges with the consolidated storage service. The objective is to create an expressive and effective performance interface between clients and server to level the ground on which the clients can optimize I/O operations and the server can efficiently provide quality I/O services. The sub-tasks of the proposed research are summarized as follows.

- **Defining Reference System as Performance Interface**

  The PI proposes the concept of *reference storage system* and uses it as a performance interface between applications and storage servers. Reference storage system can be a real or hypothetical system whose performance characteristics can be obtained and mimicked on-line to be used as a performance goal by the consolidated storage system. Example reference systems include a directly attached storage system that users had configured and optimized the I/O operations of their applications against, or a storage system on which users have run and evaluated their applications and received satisfactory performance. Without further characterization and quantification of their performance needs in terms of I/O patterns or I/O devices, they are able to simply designate a reference system to set up a *virtual*

storage device (or simplified as VSD hereafter). It is the server that is responsible to implement the VSD whose performance is the same as, better than, the reference system. The PI proposes to use the machine learning techniques to automatically derive users' QoS requirements from the reference system, and to design a systematical method to train machine learning models for high prediction accuracy.

- **Effectively Supporting Virtual Storage Devices**

  As there can be multiple VSDs hosted on a consolidated system, the PI will study several critical issues that are not seen in a dedicated storage system. First, how should the resources be allocated among the VSDs so that they have their QoS performance requirements met? In other words, we should ensure each VSD receives the service only for its required quality unless excess resource becomes available. Second, how to minimize the interference among the VSDs that co-exist on the same disk array? The third issue comes with the batching technique used to reduce interference, which leads to large performance variation. This is a difficult compromise currently practitioners have to make, which has to be removed so that the performance behaviors of a VSD are really close to its reference system. The PI proposes a multiple-clock framework to keep track of service received by each VSD and available system capability, and will design efficient scheduling algorithms to address the challenges.

- **Migrating virtual storage devices for high system efficiency**

  In a consolidated storage system consisting of different types of disk arrays, initial selection of a array to place a VSD may be based on conditions such as availability of extra storage space, the VSD's required performance, and the excess capability on the array. However, with the revelation of patterns of the requests to the VSD, such as arrival rate and spatial locality, the initial choice of a VSD placement may prove to be sub-optimal. To achieve high efficiency for the entire storage system, the PI will develop algorithms for placement and migration of VSDs among disk arrays by considering both static and dynamic information. Static information includes performance characteristics of VSDs (representing QoS requirements) and those of physical arrays. Dynamic information describes usage patterns of VSDs, which will be efficiently tracked and collected. The PI will develop methods to quantify the information to make effective migration decision.

- **System Implementation and Evaluation**

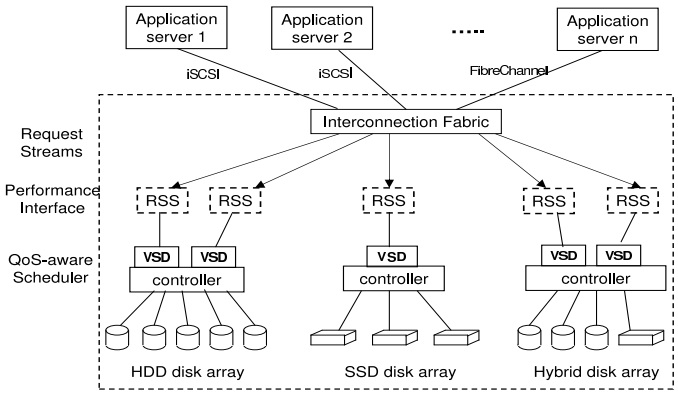


Figure 2: Consolidated storage system (in the dotted line box) are shared among multiple application servers. Performance interface is specified via reference storage system (RSS) and quantified using on-line simulators. Each VSD has its performance interface, which is implemented with an efficient QoS-aware scheduler. Decision on VSD migration is made based on characteristics of request streams, RSSs and VSDs (not depicted).

We will implement a prototype of a consolidated storage system with a user-friendly performance interface and efficient system support to guarantee QoS. Figure 2 shows the architecture of the prototype. In the implementation, we will integrate three components, namely, defining reference system, QoS-aware request scheduling to implement the interface, and VSD migration for high system utilization, in a synergistic fashion. Our initial efforts are to define reference system and implement VSD on Linux MD (Linux software RAID) as a proof-of-concept. The full-scale system implementation will provide standard block protocol access, such as iSCSI and Fibre Channel, to application servers. This effort will be in collaboration with Storage Systems Group at Seagate Research. We also plan to conduct large-scale and extensive performance evaluation on the prototype. In addition to using publicly available I/O traces and benchmarks such as those about financial transactions and Web Search [9, 11], we will also use applications and traces on high-end scientific computing from Los Alamos National Laboratory and on enterprise computing from Seagate and Yahoo! for a comprehensive testing by using both industry and national lab workloads (Collaboration letters are appended).

The rest of this CAREER proposal is organized as follows. Section 2 describes the related work. Sections 3, 4, and 5 present three major proposed research activities on the system design, namely, RSS definition, QoS scheduling, and VSD migration. Section 6 outlines projected research milestones. Section 7 describes education plan. Section 8 highlights the PI's prior research and education accomplishments. At last, section 9 summarizes prior NSF supports.

2 Related Work

As consolidated storage service promises significant benefits, performance assurance for its users is one of the most critical issues to be addressed and much research work has been focusing on it. The work related to the proposed project is discussed in the below.

2.1 Research on Performance Interfaces

The commonly used metrics for users to present the QoS requirements to storage service are throughput and response time of data access [45, 48, 71, 89, 107]. These two metrics have been widely used for providing guaranteed service for high-speed packet-switched networks [37, 38, 78, 108]. A serious issue with the use of simplistic parameters is that service quality can significantly affected by the characteristics of input traffic or workloads. To mitigate of the issue, researchers developed workload models to derive performance bounds, including deterministic traffic models [32, 33] or stochastic traffic models [30, 67, 105] in the networking domain and workload models [27, 84, 88, 91] in the storage domain. As a workload can be very dynamic, Constraints are introduced to guarantee QoS only when the constraints are met. For example, in the pClock scheduling [45], the response time of a request is bounded if the request burstiness (the number of pending requests) and request arrival rate are less than the pre-set thresholds.

However, the relationship between the imposed constraints and the guaranteed performance may not reflect the performance expectation of users. To provide flexibility, Cruz et al. introduced the notion of service curves to characterize service quality [34, 76, 78, 85] in the networking domain. With a service curve, QoS requirements can be expressed as a function of input traffic rates. Facade, a high-end storage system prototype [70] has adopted this performance interface to allow higher response times when I/O rate increases. However, it is a challenge for users to determine a series of performance bounds, each associated with an I/O rate. Furthermore, a key operation in applying the technique is to calculate deadlines for requests from a given service curve. However, not every service curve has a treatable deadline calculation [49].

Spatial locality is a unique property for the storage system and even harder than I/O rate to be included in the performance interface. So performance bounds are specified independent of sequential and random access in most systems [25, 29, 48, 45, 63, 70, 96, 107]. An exception is the Argon storage server [96], in which a client requests a fixed fraction of a server's capability, which is equivalent to setting performance

bounds aware of spatial locality. However, by binding the QoS requirements proportionally to the server's capability, a user cannot flexibly device a performance model of his choice. Furthermore, to use the method, users need to be aware of the server's total capability.

In contrast, in our proposed project users can choose any storage system that matches their performance needs as the reference system, which then serves the performance interface. In addition, mapping a workload of specific characteristics to its appropriate storage device for high service quality and system utilization is a fundamental topic in the research of storage administration [17, 19, 23, 39, 91], whose experience can provide guidance on the selection of reference system.

2.2 Research on Characterizing Storage Systems' Performance

To use a reference system as a performance interface, its performance behaviors must be well characterized to provide performance goals. For this purpose there are three methods, namely, analytic device modeling, simulation, and black-box modeling, and abundant research artifacts are available for us to leverage.

Because of their nonlinear, state-dependent behavior, building accurate analytic models of disk drives is a non-trivial task. In [77], Ruemmler and Wilkes developed analytic disk models that take into account head positioning, platter rotation, and data caching and read-ahead. The model is further fine-tuned by Worthington et al. [102], resulting in the widely used disk simulator, DiskSim [5], which represents the state of the art in disk simulation. DiskSim models almost all performance-relevant components of a disk, including device drivers, buses, controllers, adapters, caches. While disk manufacturers usually do not disclose internal technical parameters of their products, techniques have been developed to effectively extract the parameters to plug them into simulators [79, 87, 103]. There are many other disk drive models and simulators [47, 51, 66, 83, 81, 98]. Some of them were developed for their own purposes [81], for particular disk types [66], or with an emphasis on specific components [83]. While disk arrays are widely used in the high-end storage systems, a lot of research work focus on the modeling and simulation of disk arrays [21, 65, 92, 95, 94, 99]. Among the work, the Pantheon storage system simulator [99, 15] was built to support the rapid exploration of design choices of storage systems in HP AutoRAID advanced disk array technology [101] and TickerTAIP parallel RAID architecture [28]. Uysal et al. developed a composite analytic model of mid-range disk array and report its accuracy within 15% of actual measurements [92].

Compared with simulations, analytic models are much faster. However, they usually cannot capture as many details as simulators. Both approaches require human expertise on the targeted system. Thus these methods are called *white box* approach. The internal architectures of today's storage systems have become more and more complex, and may include proprietary configurations and use of algorithms and optimizations that are not disclosed and cannot be extracted with automatic approaches such as DIXTrac [79]. Therefore, building an accurate model or simulator using white box method cannot be a general solution for defining *any* given storage system as a reference system. In contrast, the so-called *black box* approach treats the system as a black box without assuming the knowledge of its internal components or algorithms. In the approach, the training data set, containing the quantified description of characteristics of input requests and their corresponding responses from the system, are recorded in a table [18, 75], fed into a statistical model [64], or a machine learning model [97, 72]. To predict a response to an input, some form of interpolation is required. Currently, the accuracy of the predictions is usually considerably lower than that of the white box method. It is recognized that the accuracy of the method relies on the selection of training data set and design of feature vectors (the set of input characteristics) [64, 97], which is currently an ad hoc process.

In spirit, the work by Popovici et al. [75] and the work by Griffin et al. [44] are closely related to our proposed research on the use of on-line simulators to guide resource scheduling. In [75], to improve disk scheduling, they train a table-based model for disk behavior, and use an on-line simulator based on the model to predict which request in the device's I/O queue has the shortest positioning time. In [44], an on-line storage emulator facilitated by DiskSim [5] is used to evaluate applications' performance on a particular storage device. In the proposed work, the PI proposes to use on-line simulator to enable a convenient performance interface and to guide resource allocation in a shared environment.

2.3 Research on QoS-Aware Scheduling

Many of existing QoS-based resource allocation strategies for storage services are derived from algorithms for allocating bandwidth and latency in the networking domain. To allocate bandwidth among traffic flows, these algorithms assign tags to requests in each flow based on their claimed bandwidths using either real time, such as VirtualClock [108] or virtual time, such as WFQ [35], WF^2Q [22], SFQ [43], and SCFQ [42]. Algorithms such as SCED [78, 34] allow control of latency of network requests independently of the allocation of bandwidth. Following the practice in the networking domain, some I/O scheduling strategies guarantee I/O service quality by tagging requests from different request streams with deadlines (or called finish times) calculated from users-specified performance bounds and estimated service times [48, 74, 45, 63, 107]. While the service time is heavily dependent on spatial locality in disk-based storage systems and the locality is not included in the performance interface, random access is usually assumed [48, 45, 74]. However, this can cause resource over-provisioning [45]. To avoid this, Stonehenge [48] allows additional streams keeps joining the system until the system is found to be overloaded. They have to use the trial and error method because the performance interface does not contain information of spatial locality and planning of resource provisioning is difficult. In contrast, the VSD proposed in our project comes with a well-defined performance interface to contain each VSD's resource consumption and to enable well-planned scheduling. The resource consumption is also contained in Argon [96] by setting explicit quanta of disk service time for each stream. However, in a shared environment it is a challenge to know to which stream a service time should be accounted [74].

Minimizing interference among co-existing virtual disks in a shared storage is a critical issue in scheduling of requests. Stonehenge [48] and Facade [70] adjust the number of pending requests for disk scheduler such as CSCAN to dispatch, to make a tradeoff between QoS and disk efficiency. Cello [82] adopts two-level scheduling strategy, coarse-gained allocation of disk bandwidth to application classes to implement QoS and fine-grained allocation to optimize disk efficiency. Argon batches a number of requests from a stream and serves them together to amortize the interference overhead [96], based on the assumption that requests to the same virtual disk are for data that are closer to each other. A solution similar to this, Anticipatory Scheduling [50], has effectively been applied in the disk scheduling with the assumption that data requested by the same process is likely to be closer to each other. In the proposed work, the PI plans to use the batching method. However, the size of the batch, or the aggregate service times of requests in a batch, is measured according to the reference system, instead of the shared physical system, to accommodate performance interface. A serious concern with the batching method is that it causes large variations of response time [96]. This is still open issue and will be addressed in our proposed project.

2.4 Research on Storage System Configuration

Research on storage system configuration is concerned with how to automatically map workloads to storage devices so that the QoS goals of the workloads are met and the storage system is well utilized [17, 19, 23, 39, 90, 91, 101, 100, 106]. An effective solution requires accurate understanding of workload characteristics such as request rate, request size, read/write ratio, and spatial locality, and of performance model of the system with different configurations. Workload characteristics are usually obtained by monitoring running of the workload on a system and analyzing the measurements from the run [19, 27, 91, 106] or provided by system administrator [17]. While it is impossible to exhaustively run every workload on all system configurations, performance is obtained through analytic model [17, 106], table-based approach [19, 91] or regression-based approach [106]. Then, a solver [19, 20] or other reasoning tools take as input the workload characteristics, performance models, and candidate system configurations to search for the optimal system design under the constraints including QoS goals and system cost. When changes of workloads or their characteristics can cause QoS goals to be violated, some corrective actions are taken, including throttling [29, 70, 91], migration [19, 101, 69], resource provisioning [17], or a combination of them [106].

There are two significant issues that need to be addressed in the aforementioned strategies to make them truly effective. One is that workload characteristics may depend on storage devices on which the workload executes (a closed-loop system) [72], which invalidates the method of evaluating a workload on different

devices assuming fixed workload characteristics. The other issue is that a device may exhibit different capability with different workloads [14, 93], which should be accounted in the placement of virtual disks. For example, a hard-drive disk shows high performance in serving sequential requests, but is inefficient in serving random requests. In contrast, a solid-state disk shows its strength with random requests, but is less cost effective to serve sequential requests.

3 Defining Reference Systems as Performance Interface.

By defining of a reference system, we mean the consolidated storage server knows how the reference storage system (RSS) would behave if a request, which is currently sent to the server, was received by the reference system, or more specifically, the server knows what the request's service time on the RSS was. For this purpose, we will develop a standardized interface for any third-party per-request service-time predictors to plug in. The predictor can be of white-box (device simulator or an analytic model) or of black box (table-based interpolation model or machine learning model). The PI will develop a suite of benchmark programs to evaluate the accuracy and efficiency of a predictor before it is deployed. While the prediction method of the machine learning models is general enough in applying in all types of systems showing its unique power in the system where the white box method cannot be applied, the PI will focus his research efforts on this method and develop a systematic approach for the application of the method.

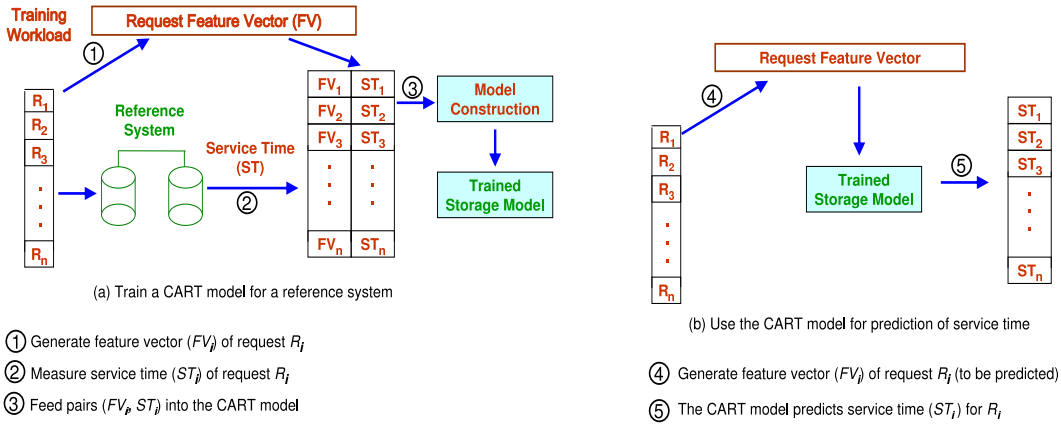


Figure 3: (a) Training a CART model for a reference system; and (b) using the model to predict service times.

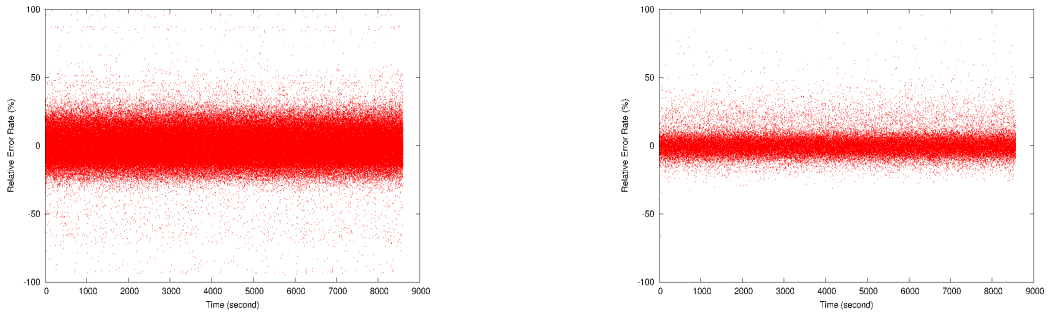
We choose to use the classification and regression tree (CART) machine-learning models [24] to obtain request service time on RSS for its efficiency and simplicity. Figure 3 shows the basic steps involved in training a model for a reference device and using the model to predict system response (which is per-request service time in the project). In the process a request's characteristics that affect its performance are abstracted in a set of measures, called feature vector, which is fed into the model with real performance measurements as training data set, and is also used as an input to probe the system for a prediction. While this method has been applied in previous work for storage performance prediction [97], its prediction accuracy is far from satisfactory. For a disk array, relative error of the predictions of per-request response time is usually well above 20%. There are several critical challenges to allow the model to make the accurate predictions.

- Feature vector must be designed to include all relevant measures. The algorithm used in the CART models makes its prediction by analyzing which of the measures in the vector have the most information with respect to the system response. An important measure misseing in the vector designed by Wang [97] is about caching effect, which makes a performance difference of several orders of magnitude. As hitting in the buffer cache is basically determined by temporal locality of accessed blocks [57], we

propose to maintain an approximate LRU stack to efficiently track recency of requested blocks and use it as a measure in the vector.

- Measures in a feature vector should have direct influence on system response but not be influenced by system response. Because a burst of requests may not immediately increase response times but accumulated requests can have long term effect on the following requests, and request rate itself is affected by response time in a closed-loop system, request burstiness or request rate should be excluded from the vector. Accordingly, we proposes to use request service time, which excludes request pending time from the response time, as system response.

- The training data set must provide a comprehensive and efficient coverage of workload characteristics. A machine learning model can make an accurate prediction only when it has "learned" similar access pattern in the training. Using a training data set covering an N-dimensional space of a vector with a fixed granularity is not feasible (either too coarse-grained or too large data set), where N is the number of measures in the vector. The PI plans to develop intelligent algorithms to explore the space with a granularity adaptive to the change of system response to achieve both high coverage and efficiency.



(A) (B)

Figure 4: The relative error rates of two trained CART models. In (A), the feature vector in [97] is used and response time is predicted. We then removed measure for request burstiness and add the measure for temporal locality in vector and predict service time (B). We use the same synthetic training data set for construction of the models for RAID5 of (4+1) Seagate ST32171W disks. The test workload the "Financial1" trace available at [11].

We have created a tool that can automatically generate a workload, for a feature vector and a storage system. Requests are selected adaptively to the observed service times of previous requests so that performance-sensitive measure is fully covered. (In the synchronous I/O, response time approximates service time). Using a trace produced by the tool we show that an optimized feature vector and system response can effectively reduce relative error rate (within 15% for 90th percentile request. Significant research is needed to enhance the tool so that it can qualitatively evaluate the relative importance of measures in a feature vector and take corrective actions to identify and remove *noisy* inputs from the training data set when the relative error rate is larger than a given threshold.

4 Effectively Supporting Virtual Storage Devices

Once a VSD is set up upon a defined reference system in a disk array, it will be implemented with other VSDs in a shared environment. To implement a performance interface of a VSD, the scheduler needs to account the system capability consumed by the VSD in serving its requests. To this end, the PI proposes a novel multi-clock framework, which sets up $N + 1$ clocks, including one reference clock (ref_clock) for each of the N co-existing VSDs, and a wall clock ($wall\_clock$) (see Figure 5(a)). A reference clock of a VSD is advanced like this. (1) When a request is selected for dispatching to the disk system, its service time

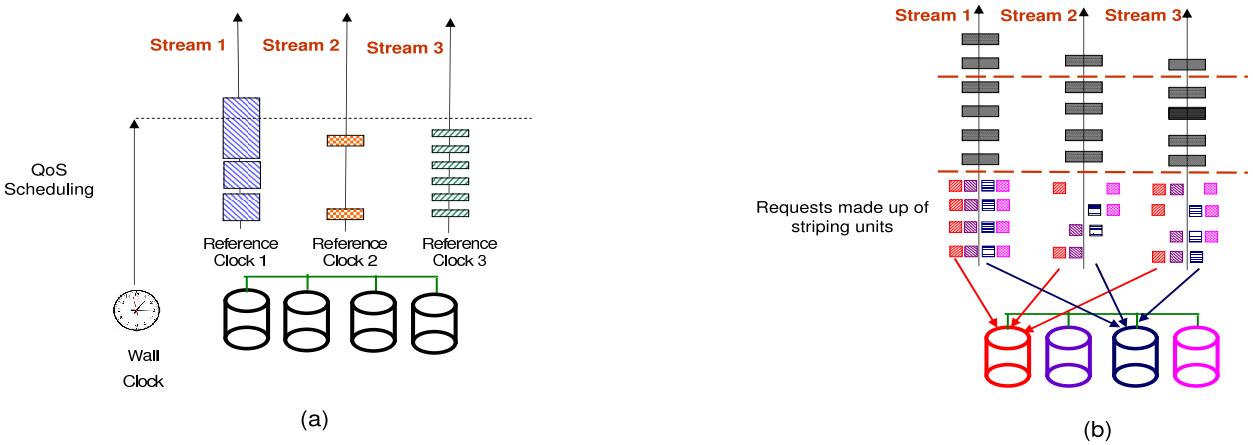
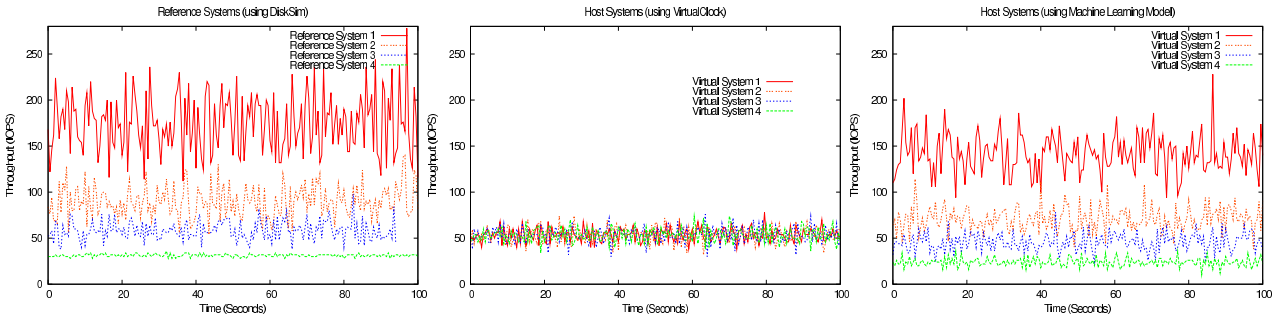


Figure 5: (a) Illustration of the proposed QoS scheduling that implements the performance interface based on defined reference system. Besides the wall clock, each VSD, serving a stream of requests from applications, has its reference clock. A request is represented by a rectangle, whose height indicates the service time of the request on the reference system. Thus, the reference clock tracks received service with respect to the reference system. Note that at a lower level each disk also has its disk scheduler, such as C-SCAN (not depicted). (b) Batching is used to minimize interference among VSDs. Requests to the same VSD are groups into of slots of fixed size (of mini-seconds), and aggregate service times of the requests in a slot on their reference system is not greater than the slot size. The performance variations due to batching are reduced by "unsyncing" the service to the striping units of a request.

($ref\_service\_time$) on the corresponding reference system is computed via the on-line simulator of the RSS associated with the VSD and we advance the clock by the service time ($ref\_clock+ = ref\_service\_time$); (2) When it is the turn of the VSD to be served but there are not any pending requests, its reference clock is set to be the wall clock time ($ref\_clock = wall\_clock$). In this way, whether or not a VSD meets its performance requirements can be easily detected by comparing ref_clock and $wall\_clock$. The scheduler chooses a VSD whose reference clock is slower than the wall clock, which indicates the VSD's performance has not kept up with that of the reference system. The gaps between the wall clock and each of reference clocks show quality of service of VSDs and tightness of resource provisioning. The significant advantage of the framework is that (1) it automatically takes all the dynamics of workload's characteristics into account, including spatial locality, I/O rate, and request size, in implementing the performance interface; and (2) it does not need to quantify physical resources consumed by each virtual disk in making the scheduling decision, which is a very challenging issue [74].

While multiple VSDs are hosted on the same physical system, their interference is a significant overhead. We plan to batch requests to the same VSD into slots and schedule requests from the same slot together (see Figure 5(b)). However, batching can lead to large performance variations because it causes a large time gap between the two consecutive services to a VSD. The PI proposes a novel scheduling to significantly reduce the variations by "unsyncing" the service to the striping units of a request. While data are usually striped over a group of disks in a disk array, requests in a VSD slot may ask for striping units belonging to different disks. Interference on a disk is reduced as long as striping units of the slot on the same disk are served together. Accordingly, a request can be divided into multiple sub-requests, each for a striping unit, whose service is unsynced. All the sub-requests to the same disk in a VSD's slot are scheduled together. Two VSDs can send their sub-requests simultaneously, as long as they are not to the same disk. In this way, multiple VSDs are served at the same time without sacrificing the benefit of batching.

As the preliminary work, we have built a simulator using DiskSim to simulate disks and disk arrays. Figure 6 plots the throughputs of four VSDs, defined on the same reference system, hosted on a disk array using the multi-clock scheduling, as well as their throughputs by using the conventional VirtualClock [108], compared with their throughputs on the reference system. The results show that the proposed scheduling



(A) (B) (C)

Figure 6: Illustration of effects of different spatial localities on scheduling strategies. The same reference system (a hard disk)is used with four workloads of a spectrum of spatial localities ("Reference System 1" indicates fully sequential access and "Reference System 4" indicates fully random). (a) Throughputs with the four workloads on the reference system; (b) Throughouts of four virtual devices that use the same SLA bounds as performance interface and are scheduled by VirtualClock [108]; (c) Throughouts of four VSDs that use the same reference system as performance interface and are scheduled by the proposed multi-clock.

strategy can faithfully implement a performance interface based on a reference system, independent of spatial locality. However, there are significant challenging issues to be addressed in the project. First, we need to design intelligent algorithms to automate the selection of slot size and make it fully adaptive to strike a balance between low interference and low performance variation. Second, the unsyncing of request service can demand a large size of buffer, especially with a large slot size. We need to find solutions to reduce it. Third, users of VSD may provide leeway on their performance requirement. For example, for throughput-conscious applications such as bulk file transfer, users may require its QoS requirements be met by average in a relatively long period. We need to develop scheduling algorithms that exploit this optimization opportunity.

5 Migrating Virtual Storage Devices for High System Efficiency

The consolidated system usually consists of heterogeneous devices, including arrays of hard drive disks, solid-state disks, or a mix of them. Each array can have different performance characteristics due to configuration differences, including performance of individual disks, number of disks in an array, array organization (RAID 1 vs RAID 5), and data striping pattern. Because VSD is created based on a reference system, its actual resource demand is bounded by the capability of the reference system. Thus, a capacity planning is possible to determine the largest of number of VSDs that can be co-hosted in a disk array with its known capability without performance violation. However, this is not necessarily translated into an efficient use of the storage system, because the efficiency relies on both device and workload characteristics.

- **Characteristics of both reference system and physical array affect the efficiency.** To quantify this effect, we define relative capability of a disk array as the number of VSDs, with a given reference system (RSS) as performance interface and a given workload characteristic, it can support. For two commonly used types of disks, HDD (Hard-Drive Disk) and SDD (Solid-State Disk [2]), spatial locality (spa_loc) and fraction of reads ($frac\_rd$) are the most relevant workload measures to characterize a workload. Therefore, the relative capability is the function $rel\_cap = f(RSS, spa\_loc, frac\_rd)$. As an example, when a hard disk is selected as RSS, rel_cap of an SDD array is 10 with random reads and 30 with sequential reads. Meanwhile, rel_cap of an HDD array is 2 with random reads and 50

[2]In this project we focus on NAND-flash-based solid-state disk, which has "the potential to revolutionize the storage system landscape" [14]

with sequential reads. So a VSD created on the RSS should be placed on the HDD array when mostly used to serve sequential reads, and on the SDD array with random reads for the highest efficiency. By normalizing the relative capability of an array over its purchase price, cost effectiveness is accounted.

- **Workload characteristics affect the efficiency.** To quantify this effect, we define workload weight of a VSD on a given workload characteristic as the ratio of the arrival rate of requests of this characteristic and the rate of requests of this characteristic that just saturates the RSS. Therefore, the workload weight is the function $wk\_weight = h(RSS, spa\_loc, frac\_rd)$. Relative efficiency is defined as $\sum_{(spa\_loc, frac\_rd)}(h(RSS, spa\_loc, frac\_rd) * f(RSS, spa\_loc, frac\_rd))$, which accumulates the weighted efficiency values on each workload characteristic point. With guaranteed performance, a VSD should be placed or migrated to the disk array which maximizes its relative efficiency.

We will design and develop tools that automatically generate workloads covering the representative characteristic points in a comprehensive and balanced fashion and collect the responses of reference systems and disk arrays to produce their relative efficiency functions. In addition, the PI will develop on-line monitoring and analysis tools to generate workload weight function for each VSD. An evaluation precess is executed periodically or triggered by substantial changes of workload behaviors or resource provisioning to compute relative efficiency of a VSD on various arrays, which determines whether a migration is carried out to improve relative efficiency and balance the load. This work includes the following significant challenges. (1) As the space of workload characteristic is large, the tools must be intelligent in its selection of interested dimension/range to explore for high efficiency. (2) The short-term workload characteristics can be highly dynamic and the migration cost of a VSD is high. It is an open question how to select an appropriate size of observation window and strike a balance between improved system efficiency and migration cost. (3) It is unknown yet how much excess capability of a disk array should be set aside to accommodate the interference overhead, which is affected by the number of VSDs and interactions among them in an array. Research is required to quantify the overhead and make it predictable to make the migration policy reliable and accurate.

6 Research Milestones

Our approach to the issues of efficiently supporting expressive performance interface consists of a combination of algorithms design, systems implementation, and performance evaluation by extensive experimentation. The research goals of the CAREER proposal over the five-year course are described as follows. **Year 1**: Design and implement a fully-functioned tool to automatically obtain training data set for the CART models. Begin evaluation of the trained models in a simulation environment. **Year 2**: Complete the design of VSD scheduling algorithms and their implementation in the Linux MD for software RAID. **Year 3**: Implement and evaluate VSDs in a block-level storage server. Begin comprehensive investigation of workload and device characteristics, as well as their interactions. **Year 4**: Design, implement, and evaluate VSD migration algorithms. **Year 5**: Comprehensive whole system evaluation using real applications and real-life traces. Write final project report, including design and implementation details, evaluation environment and results, deliverables, and motivated future work. All the source code will be placed in the public domain for wide uses, comments, and further improvements.

7 Education Plan

The research activities in the project will be carried out in parallel with a comprehensive education plan, and we will integrate the research results into both undergraduate and graduate education. Our objectives are to address some critical issues on the computer system education at Wayne State and in the ECE and computer science community of the nation by significantly improving curriculum quality and scope, including curriculum development, promoting computer system research among undergraduates, and community outreach with a focus on under-represented minorities in the Metro Detroit area.

7.1 Curriculum Development

Creation of an effective learning environment for operating system curriculum. The ECE Department of Wayne State University does not offer undergraduate-level operating system courses for several years. One reason is the low students' demand on the course. The PI has done a survey among the undergraduate students and found that the frustrating experience with experimentation of the course is a major reason, as the theory and concepts of operating systems at an undergraduate level are not difficult. As the operating system class has become highly experiment oriented, and a specific concern is related to inadequate experimental environment and project design. This is a common concern in many ECE departments that emphasize hardware design and computer architecture, as well as in many small-scale computer science departments in the nation. A widely adopted approach is to use OS simulators such as Nachos [8], which can be simplistic and lack realism. However real kernels requires immense time and probably dedicated hardware. An even bigger challenge is that most of students of Wayne State are commuter students, which makes them hard to dedicate their after-class time in an on-campus lab (There are several dozens of other universities in the nation facing the same challenge). While setting up virtual machine platforms in a server allows students to have remote access at any time to work on real kernels [73], an increasingly important ingredient that is directly related to the proposed project, the efficiency of I/O system, cannot be experimented with in a shared virtual storage environment. To address this problem, we plan to integrate the technique on time-accurate storage emulation [44] into a Linux virtual machine such as UML, in which a dedicated storage system is emulated for each student using the DiskSim simulator [5]. This will provide students with an easy to access and I/O performance-conscious virtual OS instruction lab using real kernels.

Admittedly most undergraduate students in the ECE department lack system programming experience, although they have great curiosity in the Linux kernel programming. The PI plans to adopt a step-by-step approach for a gradual learning curve to keep students' enthusiasm and make the learning an enjoyable process through carefully designed lab projects. Each project will be designed to have two parts. The instructor provides the code of Part I to ask students to patch it into the kernel. By the running the code in this part, students would be impressed by what the OS can do and be motivated to learn how it actually works. Part II requires student implement their own design into the given code so that they can appreciate their contribution. By creating a virtual Linux hacking lab and designing learning-friendly lab assignments, the PI aims to revitalize the OS teaching at Wayne State. The PI also plans to disseminate the courseware and documented teaching experiences to the community by posting them on our web site and writing paper to education journals and conferences such as SIGCSE. We believe that the proposed curriculum development will not only improve the OS teaching quality and students' learning experiences at Wayne State, but also allow other ECE and small-scale CS departments to adopt our tools and documents in their classes.

Integration of the proposed research results into the curriculum development. The PI will offer two graduate-level courses, *Storage System Design* and *Machine Learning for Networked Computer Systems*, as vehicles to get students involved in the proposed research and to benefit from its results.

In response to the high demand in today's IT market on data and storage systems, the first course will cover both fundamentals such as data organization on disks, storage interconnects, and system architectures, and advanced topics such as storage virtualization and automated storage management. Within a three-year time frame the PI will integrate a prototype of the proposed consolidated storage system into the course and assign lab projects based on different aspects of the prototype system. The course will put particular emphasis on the realization of a student's potential, instead of being oriented to students who have been ready to contribute. To this end, the PI will develop course projects so that students develop essential skills which are of immediate practical use for students who later join the proposed research. The course will also open to senior undergraduate students. By providing them with early exposure to research activities, the PI aims to culture their interest in pursuing graduate-level system research.

The second course teaches students how to develop statistical machine learning models to characterize the networked systems uncertainty and to automate resource management for adaptive, highly reliable, and self-manageable systems. The course will help students understand and solve issues involved in increasingly

complex systems from a new and promising perspective. The course teaching will be teamed with other professors at Wayne State (Prof. [redacted] and Prof. [redacted] of the ECE department and Prof. [redacted] of the Mathematics department), who have been awarded an NSF fund (DMS-0624849) to study machine learning techniques in the management of cluster systems. The PI will focus on the use of the machine learning technique to model storage systems and its workloads. This course will also serve as a forum to stimulate collaborations among faculty as well as students on the emerging research paradigm.

7.2 Promoting Computer System Research among undergraduate students

One of the challenges facing graduate-level education in computer science and engineering in the U.S is that a decreasing number of undergraduate students pursue their studies for a career development in a highly challenging technology area, specially for under-represented minority and women students. According to the most recent CRA Taulbee report, less than 15% of Computer Science Ph.D. graduates are women, and less than 4% are under-represented minorities. Furthermore, only a small percentage of female and minority Ph.D. students pursue research career in computer systems. The PI believes that efforts on actively recruiting are not enough to change the reality. A more effective and long-term solution is to promote a culture of participating computer-related research in the undergraduate students. Due to his research background, the PI plans several efforts to promoting computer system research among them in Wayne State.

- The PI plans to get involved into the Wayne State University Linux Users Group (WSULUG), that is formed by individuals of Linux fans from Wayne State and Southeastern Michigan. The PI will leverage his resource to encourage and facilitate students to learn and study the kernel designs. The PI has been maintaining a productive Linux kernel development platform, on which several important research ideas have been efficiently implemented, tested, and transferred to the Linux and BSD open source community. In addition, the PI has a close tie with memory management group for Linux kernel development (LinuxMM), where the PI has contributed some important algorithms [7]. The PI will provide technical consultation, post self-contained mini-projects on-line for motivated students to experiment on the Linux platform, guide students to evaluate Linux patches and write technical reports, and supervise undergraduate honor projects.

- The PI plans to partner with WSU-IEEE, student IEEE chapter at Wayne State, to hold regular seminars for students and faculty to share their learning and research experience, to promote closer bonds between undergraduate students and faculty. In parallel with the progress of the research project in the proposal, the PI and the graduate students will take turns to report the progress and next challenges. In the process, the PI will specifically make an extra effort to attract undergraduate students in women and under-represented minorities group to his research group. The PI also plans to invite our collaborators at national labs and industry to give guest lectures to broaden students' vision and for students to seek internship opportunities.

- To better support the undergraduate research, the PI plans to apply for an NSF Research Experience for Undergraduates (REU) supplement to this proposal if it is funded.

7.3 Community Outreach

Wayne State is located in the midtown of Detroit, in a community where many people do not have access to the regular university education and there is a great demand to increase computer literacy to improve life quality. As a member of the community, the PI will motivate the students in his research group and in his classes to donate their time and efforts along with him to teach basic computer skills in the neighborhood. There are two venues for us to initiate the outreach program. One is computer summer camp for K-12 students held in the campus of Wayne State University each summer. We will take this opportunity to demonstrate the systems in our lab and give lectures to introduce development of today's computing

technologies. The other venue is the Detroit public library next to our campus, who regularly provides computer training courses, such as the use of office software and Internet, to residents. We will volunteer to serve as instructors to make computers become valuable resource for any people. I believe these efforts also benefit our student participants, as they experience how large an impact of the knowledge they learned can be on the community. In short, it is our objective to have impacts on both research community and neighborhood community.

8 Prior Research and Education Accomplishments

Research Accomplishments This proposal will build on the PI's prior system research experience with memory management [57, 61, 62, 58, 52], performance improvement for storage system [54, 55, 60, 53, 59, 104, 36, 68], and system support for fault tolerance in parallel computing [41]. Many of the research projects involved extensive system development and have generated substantial impact in academia and industry. The expertise the PI gained in the research activities has provided the PI with a strong background with algorithms design and system implementation, which is demanded in the proposed work. Specifically, the swap-token project is focused on reduction of thrashing of memory pages to the disks. The algorithm proposed in the project [61] has been officially included in the Linux kernels [3]. As locality is one of the keys to high storage systems' performance, the PI has conducted extensive research on the analysis of locality, as well as algorithms' design and in-kernel implementations. By using reuse distance to quantify temporal locality, the PI has designed LIRS [57] and Clock-pro [52] replacement algorithms, which achieve a comparably high hit ratio of I/O requests of various access patterns. CLOCK-pro has been officially adopted in the NetBSD kernel [6]. In the DiskSeen project [36], spatial locality is well exploited to search for regularity of block access on the disk, which enable efficient disk-level prefetching. The DULO project addresses the issue of making a tradeoff between temporal locality and spatial locality so as to benefit from the high performance with sequential access on the hard disk. The buffer cache is also studied in the context networked storage systems by taking the interaction among distributed storage devices into account. The proposed project will leverage these research results, especially those on the impact of workload characteristics on the storage performance.

Education Accomplishments The PI has significantly contributed to the curriculum of the ECE department of Wayne State by introducing and improving system-area courses. In the last two years, the PI has designed two new graduate-level courses: *Computer Storage and Operating System Design* and *Caching and Prefetching Techniques in Computer Systems*. In addition to introduction of basic system design concepts and techniques in class, the PI has maintained a Linux kernel programming lab and designed lab projects on Linux kernels to help students master the contents they learned in class and gain hands-on experiences. The feedbacks of the course are very encouraging. The students' evaluations of both courses are 4.5/5, and students are so motivated that they sent a petition letter to the department chair for offering similar course each semester. The PI also taught undergraduate courses with around one third of the classes are under-represented students, including African American and female students.

9 Prior NSF Supports

The PI has been supported by two NSF grants. Among them, he is the PI of Grant CCF-0702500 ("Collaborative Research: Algorithms Design and Systems Implementation to Improve Buffer Management for I/O Data Accesses", $275K, from 6/1/2007 to 5/30/2010). In the first year of the support, major efforts are made on the characterization of workload characteristics [68, 31, 56], which partially motivated this proposed research. The second Grant CNS-0708232 ("CRI: Reconfigurable High Performance Cluster Computing and Medical Engineering Applications", $200K, from 9/15/2007 to 8/31/2009) is an equipment grant, in which he is a co-PI [redacted] is the PI). This grant allows us to create and maintain a research infrastructure for cluster computing and storage studies, which will be used for this proposed research.