The Server Reassignment Problem for Load Balancing in Structured P2P Systems

Chyouhwa Chen and Kun-Cheng Tsai

Abstract—Application-layer peer-to-peer (P2P) networks are considered to be the most important development for next-generation Internet infrastructure. For these systems to be effective, load balancing among the peers is critical. Most structured P2P systems rely on ID-space partitioning schemes to solve the load imbalance problem and have been known to result in an imbalance factor of $\Theta(\log N)$ in the zone sizes. This paper makes two contributions. First, we propose addressing the virtual-server-based load balancing problem systematically using an optimization-based approach and derive an effective algorithm to rearrange loads among the peers. We demonstrate the superior performance of our proposal in general and its advantages over previous strategies in particular. We also explore other important issues vital to the performance in the virtual server framework, such as the effect of the number of directories employed in the system and the performance ramification of user registration strategies. Second, and perhaps more significantly, we systematically characterize the effect of heterogeneity on load balancing algorithm performance and the conditions in which heterogeneity may be easy or hard to deal with based on an extensive study of a wide spectrum of load and capacity scenarios.

Index Terms—Distributed hash table, load balance, local search, structured peer-to-peer system, generalized assignment problem.

1 Introduction

Peer-to-peer (P2P) systems make it possible to harness resources such as the storage, bandwidth, and computing power of large populations of networked computers in a cost-effective manner. In structured P2P systems, data items are spread across distributed computers (nodes), and the location of each item is determined in a decentralized manner using a distributed hash lookup table (DHT) [1]. Structured P2P systems based on the DHT mechanism have proven to be an effective design for resource sharing on a global scale and on top of which many applications have been designed such as file sharing, distributed file systems [2], real-time streaming, and distributed processing.

In these systems, each data item is mapped to a unique identifier ID drawn from an identifier space. The identifier space is partitioned among the nodes so that each node is responsible for a portion of the ID space, called zone, and storing all the objects that are mapped into its zone. However, there are several problems with this approach. Following [3], let N be the number of nodes in the system and $f_{\max}$ be the ratio of the largest zone size to the average zone size. Then, it is known that $f_{\max}$ is $\Theta(\log N)$ with high probability [3], [4]. Therefore, this could result in a $\Theta(\log N)$ load imbalance factor in the number of objects even when nodes have homogeneous capacities. In practice, the resources of P2P systems are most likely overlaid on top of peer nodes with extreme heterogeneity in hardware and software capabilities. Some peers may be large servers with plenty of computing power and large storage access through a reliable and high-speed network, whereas other peers may be handheld devices with wireless connections that have limited storage, computing power, and unreliable connections. Although there are approaches that are effective in partitioning the ID space according to node capacities [5], these approaches cannot adapt to dynamic workload changes in real networking conditions.

Therefore, we focus on the virtual server (VS) or migration-based approach to load balancing in this paper, which can move portions of the load off an overloaded physical node dynamically and has already been explored in a number of studies [6], [7], [8], [9], [10]. A VS looks like a peer in the original DHT architecture for being responsible for a zone, but each physical node may be associated with several VSs. Even though the VS mechanism increases the path length on the overlay, it offers a vehicle to move a load from any physical node to any other physical node, which is crucial for load balancing. Several points are noteworthy in the VS concept. First, the transfer of a VS from one physical node to another is equivalent to a leave followed by a join operation to the underlying DHT and is supported by the DHT framework. Second, the concept is applicable to many types of resources such as storage, CPU processing time, bandwidth, etc. However, we deal with load balancing P2P systems for storage objects only because this type of application is important in practice [2], [11], [12] and other load balance objectives may require completely different optimization formulations.

Following Rao et al. [6], let t_j denote the target load of physical node j, l_i denote the current load on VS i, I denote the set of all VSs, and J denote the set of nodes in the system. A P2P system is defined to be balanced if the sum of

- *C. Chen is with the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, #43, Sec. 4, Keelung Rd., Taipei, 106, Taiwan, ROC. E-mail: cchen@csie.ntust.edu.tw.*
- *K.-C. Tsai is with the Institute for Information Industry, #7 Fl.-1, No. 6, Lane 44, Wanan Street, Taipei 116, Taiwan, ROC. E-mail: garytsai.tw@yahoo.com.tw.*

Manuscript received 30 Oct. 2005; revised 4 Mar. 2007; accepted 20 June 2007; published online 5 July 2007.
Recommended for acceptance by K. Hwang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0457-1005.
Digital Object Identifier no. 10.1109/TPDS.2007.70735.

the load s_j of a physical node j is smaller than or equal to the target load of the node for every node $j \in J$ in the system. That is,

$$s_j = \sum_{i \in I} l_i x_{ij} \leq t_j, \forall j \in J. \quad (1)$$

In (1), the binary variable $x_{ij} = 1$ indicates that VS i is assigned to node j. A physical node j is called heavy if its combined load exceeds its target load, $s_j > t_j$; otherwise, the node is called light. When the system is imbalanced, the goal of a load balancing algorithm is to find a way to move VSs from heavy nodes to light nodes in a way that minimizes the total load moved. In [6], [7], [8], [9], and [10], various heuristic schemes have been proposed and appear to achieve good performance. In light of the importance of this issue, a more systematic treatment of the problem is desirable. Therefore, we opt for an optimization-based approach and try to leverage the vast amount of research performed for solving similar problems, in particular, the general assignment problem (GAP) [13], [14], [15], [16], [17], [18], [19], [20]. GAP considers the minimum cost assignment of tasks to agents such that each task is assigned to one and only one agent subject to capacity constraints on the agents. A problem instance has the following inputs: a set of tasks I, a set of agents J, a set of constants l_{ij} denoting the incurred load when task $i \in I$ is assigned to agent $j \in J$, a set of parameters c_{ij} as the cost incurred when task i is assigned to agent j, and t_j as the capacity of agent j. The constraint is that each task i can be assigned to exactly one agent j without exceeding the capacity of the agent. The formulation of the integer programming GAP is shown as follows:

$$\text{Minimize } f(x) = \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \quad (2)$$

$$\text{s.t.} \quad \sum_{i \in I} l_{ij} x_{ij} \leq t_j, \forall j \in J, \quad (3)$$

$$\sum_{j \in J} x_{ij} = 1, \forall i \in I, \quad (4)$$

$$x_{ij} = \begin{cases} 1, & \text{if task } i \text{ is assigned to agent } j, \\ 0, & \text{otherwise.} \end{cases} \quad (5)$$

The objective of GAP is to find a solution that minimizes the total cost, as defined in (2). Equation (3) enforces the resource capacity constraints of the agents. Equation (4) guarantees that each task is assigned to one and only one agent.

The standard GAP formulation as stated above cannot be applied to the load balance problem directly. The load balance problem must start from an infeasible assignment in which the workload on some nodes exceeds their targets and find a feasible assignment while minimizing the total amount of load moved. We will formulate the P2P load balancing problem in terms of a GAP. Let x_{ij} be the decision variable that indicates if VS i is assigned to node j. Since we need to represent the system state before movement, a new state-independent constant y_{ij} is introduced, which takes on a value of 1 if VS i is originally stored on node j in the starting assignment and 0 otherwise. Note that accessing this state-independent constant involves a simple memory reference. Following [6], the cost c_{ij} incurred when a VS i is moved to a node j is defined to be equal to the workload l_{ij} of i when i is not originally on j, that is, $y_{ij} = 0$. However, $c_{ij} = 0$ in the case i is moved back to node j if i is originally stored on node j, that is, $y_{ij} = 1$. As moving i to any node other than its original node incurs the same cost, to simplify the notation, a new constant $l_i = l_{ij}$ is introduced. Combining y_{ij} and c_{ij} gives a concise definition of cost c_{ij}:

$$c_{ij} = l_i(1 - y_{ij}), \forall i \in I, j \in J. \quad (6)$$

The problem of load balancing in a P2P system can then be transformed into a GAP as follows:

$$\begin{aligned} &\text{Minimize } f(x) = \sum_{i \in I} \sum_{j \in J} l_i(1 - y_{ij}) x_{ij} \\ &\text{s.t. } (3), (4), \text{ and } (5) \text{ holds.} \end{aligned} \quad (7)$$

The first major contribution of this paper is to derive an effective server reassignment algorithm for the solution of the load balance problem and address issues such as node registration policies and the effect of the number of directories, which are vital to the success of the framework. Our second major contribution is to systematically investigate the effect of forms of heterogeneity on the difficulty of the server reassignment problem. Work in P2P load balancing increasingly employs heavy-tailed distributions such as power-law, Zipf's law, and Pareto distributions [21], [22] to characterize workload and node capacity in their studies because these distributions are considered to represent the essence of heterogeneity and are hard to deal with due to their high variability [6], [7], [8], [9]. For example, in [6], the power-law exponent $\alpha = 3$ is used, where the variance is infinite. The authors mentioned that this setting is a particularly bad case for load balancing. In [9], the Zipf parameter range values between $0.8 \sim 2.4$ are used to investigate the effect of workload skew on system performance, corresponding to power-law exponents $2.25 \sim 1.42$. In [8], power-law exponent values from 1.5 to 3.5 are used in their simulation studies. These and other works consider distributions with infinite variance to represent the hardest cases to solve and focus on a workload exhibiting such characteristics. However, we found that such settings are not the most difficult ones. Our investigations provide some results on the hardness of the server reassignment problem in general. Therefore, the second major contribution of this paper is a systematic evaluation of our and other related algorithms in a wide range of settings to understand their performance as the degree of heterogeneity varies.

The remainder of this paper is structured as follows: In Section 2, related works in GAP and structured P2P systems are briefly discussed. Our proposed algorithms are presented in Section 3. Results from an extensive set of experiments to compare our proposals with others are presented in Section 4. Finally, Section 5 concludes the paper.

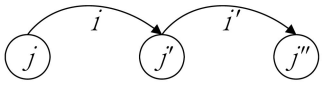

Fig. 1. Shift the task i to agent j' and i' to agent j'' simultaneously.

2 Related Works

The load balance problem for heterogeneous overlay networks has attracted much attention in the research community only recently. This paper focuses on proposals based on the notion of VSs [2], whose explicit definition and use for load balance were first proposed by Rao et al. [6]. In [6], the authors also introduced several load balancing algorithms, including many-to-many with dislodge (called M2M in this paper) and M2M without dislodge (called DM2M), based on the assistance of a new type of peer node, called directory. Load balance actions are executed by VS reassignment algorithms executed on directory nodes. The performance of M2M is shown to be superior to DM2M. Both algorithms are intuitively appealing, but there are several important issues with the general M2M approach. First, the algorithms may not be able to find feasible assignments in certain very simple situations. The problem is that, starting from a set of VSs S, the M2M strategy searches only in the direction of decreasing total load in S. Presumably, this strategy guarantees algorithm termination. Second, a number of design issues, such as how nodes should register with directory nodes, are not adequately addressed. Later, in [8], a clustered VS scheme is presented that can be viewed as an optimization of the basic VS framework to reduce the overhead involved in the VS framework. However, VSs cannot be moved and, therefore, the scheme cannot respond to dynamic changes in network conditions.

We next briefly describe related work in GAP. Sahni and Gonzalez [23] proved that the problem of deciding if there exists a feasible solution is NP-hard. Later, Fisher et al. [14] proved that the generalized assignment problem is NP-complete. Notable existing approaches for solving GAP include [15], [16], [17], [18], [19], and [20]. Of particular interest to us is the work done by Lourenco and Serra [20], who proposed a general framework with basic elements extracted from previous works [24], [25]. Lourenco and Serra's general strategy is also adopted in our algorithm, albeit with much modification due to the much larger search space for our problem.

The general strategy in the GAP proposals is the creation of a solution and local searching in the "vicinity" of the created solution before moving on to the next iteration. Therefore, the notions of *neighborhood*s and *move*s for conducting a local search with respect to a created solution are needed. With respect to a solution, *a shift move* consists of removing a task (VS in our case) from one agent (node) and assigning it to another, whereas a shift neighborhood is a set of such moves. Similarly, *an ejection-chain move* is a compound sequence of one or more shift moves. The length of an ejection-chain move is the number of shift moves in the sequence. An ejection-chain move of length 2 (two shift moves) is illustrated diagrammatically in Fig. 1. The figure illustrates the action of removing a task i from an agent j, assigning i to a different agent j' and then removing a task i' from agent j', assigning i' to an agent j''.

Clearly, each move leads from one solution to the creation of another by changes in the selected VS assignment to nodes. Fig. 1 includes the special cases single-step shift moves when i' is not removed and swap operations when j'' is again equal to node j. A collection of ejection-chain moves is called a *neighborhood*.

We note several features in the neighborhood search strategies employed in [20]. First, Lourenco and Serra employed both a simple shift neighborhood and a restricted ejection chain of length 2 neighborhood. Both neighborhoods are searched using a single cost function, making no distinction between feasible and infeasible solutions. The fundamental insight of this paper is that a more focused definition of the neighborhood search spaces for the local search is needed. When the goal is to make an infeasible solution satisfy the capacity constraints, a large number of the neighbors can be ruled out quite simply by considering moves that lessen the violated capacity constraints. Similarly, when the goal is to reduce the cost of an already feasible solution, those moves not conducive to cost reduction do not need to be considered. We propose to classify the search spaces into two types and develop a corresponding search strategy as an effective approach to tackle the problem. The details are explained next.

3 Proposed Algorithm

The overall algorithm we propose is called dual-space local search (DSLS) and is based on the framework by Lourenco and Serra [20]. DSLS is an iterative procedure with three main steps performed in each iteration. First, an initial solution is generated using the ant system heuristic (ASH) algorithm. The solution is then improved as much as possible to reach its local minima using the descent local search (DLS) algorithm. The pheromone variables are then updated and the overall procedure is executed again.

The rationale for the design is that, even though ASH is an effective randomized restart procedure that can construct a good initial solution in a reinforcement style of learning using the pheromone trails, it is not as effective for finding nearby local optima solutions several steps away from the generated solution. A local search algorithm must be used to improve the constructed solution to enhance the search in terms of earlier detection of high-quality solutions. Our DLS algorithm is the local search component that finds the local optima solution in the neighborhood of a given initial solution. In addition, we distinguish between the search spaces by the purpose they are to achieve and thus can search in significantly smaller search spaces. The DSLS procedure is given as follows:

- **Construction.** The algorithm invokes the ASH algorithm to construct an initial solution x for the current iteration.
- **Improvement.** The algorithm then invokes the DLS procedure to derive a local minimum solution based on the initial solution. DLS is another iterative loop

comprising two main phases for searching in the two ejection-chain neighborhoods:

- First, if the initial solution generated in the first step is not a feasible solution, the algorithm performs a local search procedure in a feasibility-improving ejection-chain neighborhood $N(x)$ to adjust the solution to a feasible one x'.
- Second, based on the feasible x', the algorithm performs another local search in a cost-reducing ejection-chain neighborhood $N'(x)$ to adjust x' to a lower cost one x''.

- **Pheromone trail update.** The current best solution will be replaced by x'' if it is better than the current best solution. The pheromone trails will be updated to reflect the effect of x''.

3.1 The Ant System Heuristic

ASH is an instance of the Ant Colony Optimization approach [26]. The input parameters are the pheromone trail variable τ_{ij}, denoting the intensity of the desire to assign task i to node j. Following the proven design in [24], our ASH assigns VSs or tasks in GAP terminology to nodes in decreasing order of the VS load. Let i be the task to be assigned:

- With probability p_0, task i is assigned to the node j^* with sufficient residual capacity and has a maximal value of τ_{ij}.
- With probability $1 - p_0$, task i is assigned to a node with sufficient residual capacity according the following probability function:

$$p_{ij} = \frac{\tau_{ij}}{\sum_{j' \in J} \tau_{ij'}}, \forall j \in J. \tag{8}$$

- If all nodes are fully occupied, the assignment is random.

The parameter p_0 controls the degree of exploitation and exploration of an ant. Exploitation causes the system to make use of its current achievement, whereas exploration forces the system find a new and better solution. The trade-off between exploration and exploitation is a key issue of an ant system. The ant exploits the path with a maximal value of τ_{ij} with probability p_0; otherwise, it explores other paths according to (8). With a high value of p_0, the best possible choice is made most often. If the probability $p_0 = 1$, the ant will always choose the best possible path and will not perform any exploration. The probability p_0 must be designed to balance exploitation and exploration. The definition for the probability p_0 in our algorithm is defined to be $p_0 = ((|I| - |J|)/|I|) \times 0.8$ [24], which has proven to be robust for GAPs.

Following Lourenco and Serra's experiments [20] for GAP, the pheromone trail variables τ_{ij} of our ASH are initialized as $\tau_{ij} = 1/c_{ij}$ if $c_{ij} \neq 0$. Due to the special properties of load balancing for a P2P system, no cost will be incurred if a VS is assigned to the originally stored node. In this case, the value of $1/c_{ij}$ would be infinite, so we let τ_{ij} be $\tau_{\max}$ if $c_{ij} = 0$, where $\tau_{\max}$ is the maximum intensity of the pheromone on a trail. Moreover, we limit the minimum and maximum of the intensity of the pheromone trail to $\tau_{\min} = 0.1 \times \min_{\forall i,j,c_{ij} \neq 0}(1/c_{ij})$ and $\tau_{\max} = |I| \times \max_{\forall i,j,c_{ij} \neq 0}(1/c_{ij})$ for updating pheromone falls inside the interval.

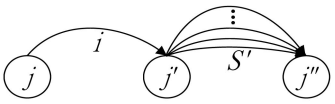

Fig. 2. Shift a VS i to node j' and S' to node j'' simultaneously.

3.2 Two-Stage Descent Local Search Procedure

DLS is itself an iterative loop whose body consists of two main phases for searching in two ejection-chain neighborhood search spaces. First, a feasibility-improving neighborhood $N(x)$ is generated to search for a feasible solution in the vicinity of x. Second, a cost-reducing neighborhood $N'(x)$ is generated to find lower cost solutions in the vicinity of a feasible solution. We next present the two ejection-chain neighborhoods and their search strategies.

3.3 Dual Neighborhood Search Spaces

The simple overflow function $f'(x)$, as shown in (9), is used to define the search spaces. The definitions of i, j, I, J, t_j, l_{ij}, and x_{ij} are the same as those in (2), (3), (4), and (5). This function measures the extent of capacity violation and is zero for a solution x if it is feasible:

$$f'(x) = \sum_{j \in J} \max\left(0, \sum_{i \in I} l_{ij} x_{ij} - t_j\right). \tag{9}$$

If an initial solution x is not feasible, that is, $f'(x) > 0$, the feasibility-improving ejection-chain neighborhood $N(x)$ is defined to be the tuples or moves (i, j, i', j', j''):

$$N(x) = \left\{ (i, j, i', j', j'') \middle| \begin{array}{l} x_{ij} = 1, x_{i'j'} = 1, s_j > t_j, \\ s_{j'} < t_{j'}, i \neq i', j \neq j' \end{array} \right\}, \tag{10}$$

where $s_j = \sum_{i'' \in I} x_{i''j}$, $s_{j'} = \sum_{i'' \in I} x_{i''j'}$, i, $i' \in I$, and j, j', $j'' \in J$.

Referring to Fig. 1 again, each tuple (i, j, i', j', j'') denotes the condition in which VS i is on node j, VS i' is on node j', node j is overloaded $(s_j > t_j)$, and node j' is not $(s_{j'} < t_{j'})$. Each tuple denotes a potential useful move, moving a VS i from an overflowing node j to another node j' with spare capacity and moving a VS i' from the node j' to any other node j''. Two local search strategies are investigated:

- The first strategy is an exhaustive approach where all moves in $N(x)$ are evaluated.
- The second strategy is a greedy approach that essentially keeps reducing the load of the currently most loaded node among all heavy nodes by shifting its lightest VS appropriately. Focusing on shifting the lightest VS reduces the total amount of load moved.

The greedy procedure is given as follows in more detail, as illustrated in Fig. 2. Let the current heaviest node be j and i be the lightest VS on j. We try to find a pair of ejection-chain destinations (j', j'') in decreasing order of

their residual or unused capacity that can be used to reduce the load on j. If j' has enough residual capacity to accept i, i is simply shifted to j'. Otherwise, an extended ejection-chain move is performed, which shifts i to j' and shifts the smallest set S' of VSs on j', whose combined load is smaller than i and whose removal would make j' light without overloading j''. This multistep move is equivalent to multiple ejection-chain moves that deal with single VSs but is more efficient because it does not require multiple outer loop iterations.

We next define the cost-reducing neighborhood $N'(x)$, which is a very particular type of neighborhood, as defined in (11). Essentially, $N'(x)$ only considers moving VSs back to their original nodes if possible. Note that this type of move will always result in a reduction of the total load moved for any given solution. Clearly, there are other more elaborate neighborhood search spaces for cost reduction. However, more elaborate search spaces also mean more computation time required for searching:

$$N'(x) = \left\{ (i, j, i', j', j'') \,\middle|\, \begin{array}{l} x_{ij} = 1, x_{i'j'} = 1, y_{ij} = 0, \\ y_{ij'} = 1, i \neq i', j \neq j' \end{array} \right\}, \quad (11)$$

where i, $i' \in I$, and j, j', $j'' \in J$.

Each tuple in (11) denotes the action of moving a VS i to its original node j' and moving a VS i' from the node j' to another node j''. Searching in $N'(x)$ is done in a manner similar to searching in $N(x)$ with an exhaustive or a greedy approach.

3.4 Update the Current Best Solution and Pheromone Trail Variables

The current best solution x^* so far will be replaced by the solution x produced in this iteration if it is found to be better. A solution is considered to be better if it satisfies one of the following two conditions:

- $f'(x) = 0$, $f'(x^*) = 0$, and $f(x) < f(x^*)$: Both the new solution x and the current best solution x^* are feasible $(f'(x) = 0, f'(x^*) = 0)$ and the cost of the new solution is lower than that of the current best solution.
- $f'(x) < f'(x^*)$: The overflow amount in the new solution is smaller than that of current best solution.

Finally, the pheromone trail variables will be updated using the following:

$$\tau_{ij}^{new} = \max\Big(\tau_{\min}, \min(\tau_{\max}, \rho\tau_{ij}^{old} + \delta \times \tau_{\max} \times x'_{ij})\Big), \quad \forall i \in I, j \in J. \quad (12)$$

The parameter ρ, $0 \leq \rho < 1$, is the ratio of the persistent pheromone trails and is set to $\rho = 0.75$. That is, $\tau_{ij}^{old}(1-\rho)$ of the pheromone evaporates in each iteration. The amount of pheromone is increased by the current local optimal solution by $\delta \times \tau_{\max} \times x'_{ij}$, where $\delta = 0.05$ if x is a feasible solution; otherwise, $\delta = 0.01$, with the restriction that $\tau_{\min} \leq \tau_{ij} \leq \tau_{\max}$. In other words, if the solution is feasible, the relevant pheromone trail variables get a bigger increment, making these trails be preferentially considered in the future. Again, these values for δ follow the recommendations in [24]. In general, the termination condition of the loop in DSLS is a fixed number of iterations. However, we also find that the performance improvement for a further search in terms of both the success ratio and the moving cost is quite small after failing to find a better solution in an iteration. Therefore, we also terminate the search at the first time a solution cannot be improved.

4 Performance Evaluation

In this section, we present the results from an extensive set of experiments to investigate the performance of the proposed DSLS algorithm in comparison with the static M2M algorithm [6] and DM2M [7] across a wide spectrum of distribution parameter values for VS load and node capacity. DM2M is the same as M2M except that the first stage of the main loop of the M2M algorithm is executed only once and the second stage of the algorithm is omitted. Exclusion of the second stage may be justified in situations where a slightly higher node utilization level is acceptable, to be reduced by later rounds of algorithm execution in a dynamic environment.

The performance evaluation experiments are presented in the following order: First, we focus on characterizing the performance in a static single-directory setting, where all possible degrees of heterogeneity at the VS workload and node capacity level are explored. Second, multiple-directory experiments in a static setting are presented. Finally, we characterize the performance in a dynamic storage-oriented dynamic setting patterned after the same application setting in [7]. Static experiments study the impact of all possible *snapshots* of heterogeneity conditions on algorithm performance and do not imply actual implementation suggestions. Single-directory experiments are useful because they help understand the multidirectory case. In the dynamic experiments, storage objects arrive and depart, resulting in the workloads on the VSs. Note that nodes do not experience churn. Besides being interesting in their own right, the dynamic experiments serve to verify the results in the static experiments.

To create the whole spectrum of settings from extreme heterogeneity to homogeneity in our experiments, the workloads and node capacities are both modeled by the Pareto distribution. Recall that an infinite variance occurs when the shape parameter value is 2 for the Pareto distribution and that the variance of capacities decreases as the shape value increases. To generate the distribution with a variance from ∞ to 0, the shape parameters α of the Pareto distribution are set in the range between a minimum of 2 and a maximum of ∞. To maintain a fixed mean μ, the scale parameter β values are then calculated according to the Pareto distribution's formula $\mu = \alpha\beta/(\alpha - 1)$ [27]. To improve the clarity of presentation, we shall avoid using Pareto-specific jargon such as "as variance increases" or, equivalently, "as shape value decreases." We will instead use expressions such as "as the degree of heterogeneity becomes more heterogeneous" or "as it becomes more heterogeneous." Similar comments apply to the homogeneous case.

In some experiment settings, problem instances may not have feasible solutions. In addition to evaluating the

TABLE 1
Parameters for Performance under Capacity and Load Heterogeneity for the Static Experiments

Parameters	Value range
Number of nodes	128
Number of VSs per node	5, following [6]
α_c : shape of node capacity	$2.0 \sim \infty$
β_c : scale of node capacity	$3.125 \sim 6.25$
α_w : shape of VS workload	$2.0 \sim \infty$
β_w : scale of VS workload	$0.5 \sim 1.0$
System utilization	0.8

performance of the algorithm on all the instances, it is desirable to evaluate them when only feasible instances are considered. However, deciding if an instance has any feasible solutions is NP-hard, as mentioned before. Therefore, we define the following *necessary* conditions. We call the instances satisfying these conditions to be *admissible* instances. Clearly, instances that have feasible solutions must be admissible but not vice versa:

- The total workload should be smaller than or equal to the total capacity, $\sum_{i \in I} l_i \leq \sum_{j \in J} t_j$.
- The maximum workload of the VSs should be smaller than or equal to the maximum capacity of the nodes, $\max_{i \in I}(l_i) \leq \max_{j \in J}(t_j)$, to ensure that at least one node capable of storing the largest VS exists.

The following are the main performance metrics used in this paper:

- *The admissible property under all workload and node capacity heterogeneity settings.* This property gives a basic understanding of the impact of heterogeneity on the problem space.
- *The success ratio of problem instances solved among admissible or all problem instances.* These experiments characterize the problem-solving abilities of the algorithms.
- *The 99.9th percentile node utilization among all problem instances.* The 99.9th percentile node utilization is the maximum over all simulated times of the 99.9th percentile of the utilizations among the nodes.
- *The total workload moved as a fraction of the total system workload among all problem instances.* These experiments characterize the overhead required to achieve load balancing.

For the static experiments, unless stated otherwise, the parameter values for workloads on VSs and capacities for nodes are as in Table 1. Values for dynamic experiments will be shown later.

Symbols α_w and β_w denote shape and scale parameters for the workload, whereas α_c and β_c are for capacities. We first fix the mean of the per-VS workload at 1. When there are five VSs on a node, the mean of the capacity per node is calculated to be 6.25 to obtain the fixed 0.8 system-level utilization. To generate the heterogeneity for both workloads for VSs and node capacities, their shape values α are varied from 2 to ∞ so that their variance varies from ∞ to 0. For each α value, a corresponding β is calculated to maintain the fixed mean.

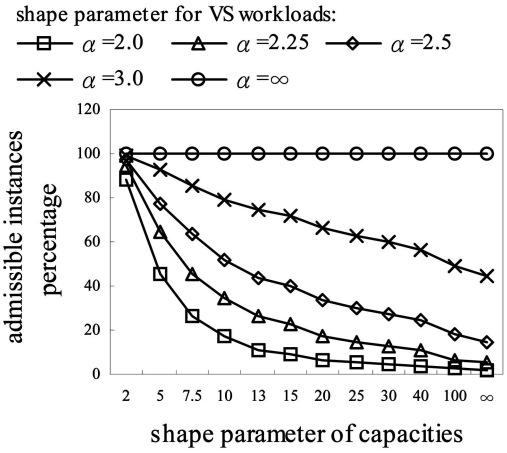

Fig. 3. The admissible property of the problem instances.

4.1 Impact of Capacity and Load Heterogeneity on Admissibility in a Static Environment

The overall impact of heterogeneity on the problem space itself is evaluated first by examining the admissibility property of the problem instances. Intuitively, when the workload exhibits more heterogeneity, it may be hard to fit them into nodes whose capacities are relatively homogeneous. In those cases, feasible solutions may exist less frequently. Fig. 3 confirms the intuition.

The x-axis represents the degrees of node capacity heterogeneity, which becomes progressively more homogeneous toward the right. Each curve depicts the percentage of admissible instances with a fixed degree of workload heterogeneity. For a fixed capacity α_c value, there are fewer admissible instances as the workload becomes more heterogeneous, as demonstrated from the topmost curve to the bottommost curve. When the node capacity exhibits extreme heterogeneity, many feasible solutions exist and the success ratio is important for algorithm performance evaluation. On the other hand, when the node capacity exhibits extreme homogeneity, node utilization is a more appropriate metric. A previous study of M2M concentrated on evaluating node utilizations in the extreme heterogeneous node capacity case [7].

When $\alpha_w = \infty$, the admissible percentage appears to be 100 percent across the spectrum of all node capacity shape values in Fig. 3. This is counterintuitive because it seems impossible for all the nodes to be large enough to hold the homogeneous workloads when the node capacity is highly heterogeneous. This is because, in a Pareto distribution, all samples generated are larger than the scale parameter value. For $\alpha_c \geq 5$ and the mean of the node capacity fixed at 6.25, the corresponding scale parameter values β_c are larger than or equal to 5. Thus, the node capacities generated can hold all the VS workloads, whose combined total workload is exactly 5.

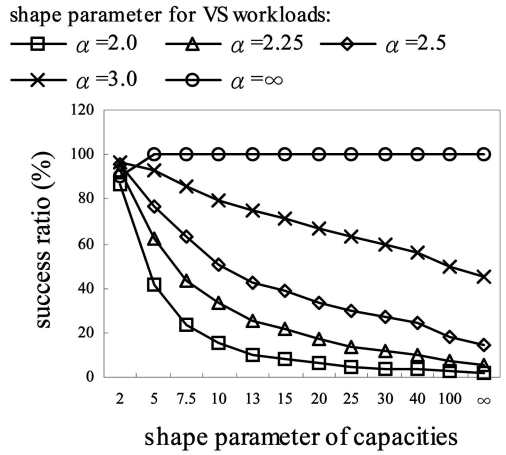

Fig. 4. The success ratios of DSLS versus the shape of capacities for all instances.

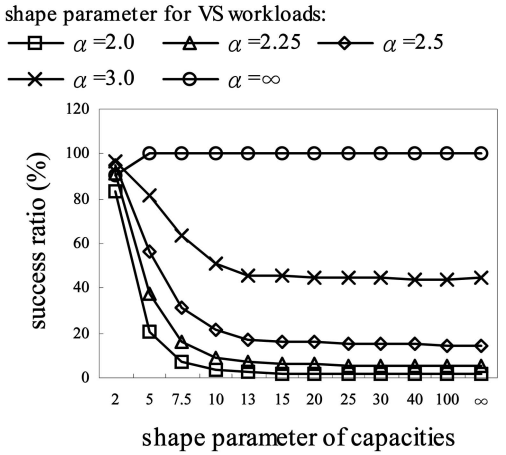

Fig. 5. The success ratios of M2M versus the shape of capacities for all instances.

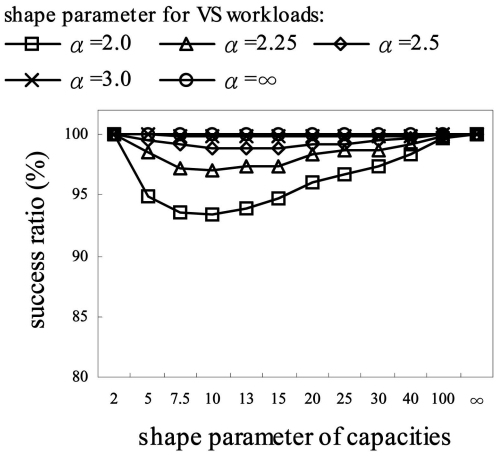

Fig. 6. The success ratios of DSLS versus capacity heterogeneity for admissible instances.

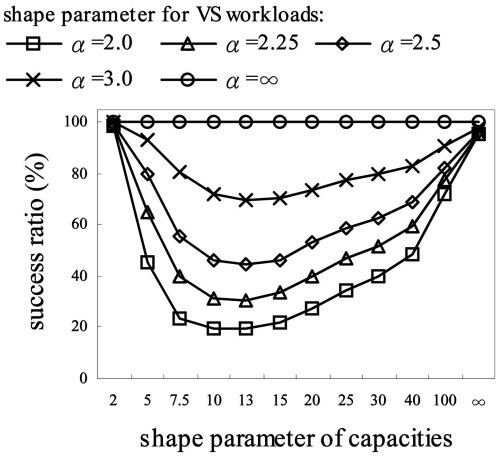

Fig. 7. The success ratios of M2M versus capacity heterogeneity for admissible instances.

4.2 Single-Directory Success Ratio Performance in a Static Environment

The success ratio performance of DSLS and M2M for all instances is depicted in Figs. 4 and 5, respectively. Clearly, DSLS performs better than M2M. To gain a better insight into the result, it is easier to examine the performance when only admissible instances are considered.

Figs. 6 and 7 depict their performance for admissible instances only. Note that both algorithms exhibit the same general trend as the degree of heterogeneity in capacity varies, achieving the best performance at the two extreme end points on the x-axis. Consistent with the findings in Fig. 3, both algorithms find the most difficulty when the workload becomes more heterogeneous, as illustrated by the curves from the top to the bottom. Again, when $\alpha_w = \infty$, the topmost curve is completely flat at 100 percent beyond $\alpha_c \geq 5$ because all generated instances are automatically feasible due to the definition of a Pareto distribution.

At the extreme left on the x-axis in both figures, both algorithms achieve the highest success ratios. This is because, in this setting, all light nodes have very large capacities and all heavy nodes have very small capacities. Any strategy that simply moves the VSs from heavy nodes to light nodes would be effective for solving the problem. Toward the center of the x-axis, the success ratios of both algorithms decrease. This is because there are fewer nodes with extremely large capacities, and the problem becomes a true generalized assignment problem. The situation is particularly severe for M2M, whose success ratio can plunge to as low as 20 percent for some combination of capacity and workload shape values. This is because M2M essentially just moves the VSs from heavy nodes to light nodes, and the second stage of the M2M algorithm is ineffective. The rightmost end of the horizontal axis corresponds to the homogeneous capacity case, which appears to be relatively easy to deal with among admissible instances.

For admissible problem instances, capacity homogeneity does not necessarily imply difficulty for the algorithms. The particular $\alpha_c = 10$ seems to be the hardest scenario to solve and will be called *the most difficult scenario* and used as a stress test in various later experiments.

4.3 The 99.9th Percentile Node Utilization Performance for All Instances

The 99.9th percentile node utilization experiment characterizes the overload situation when all instances are

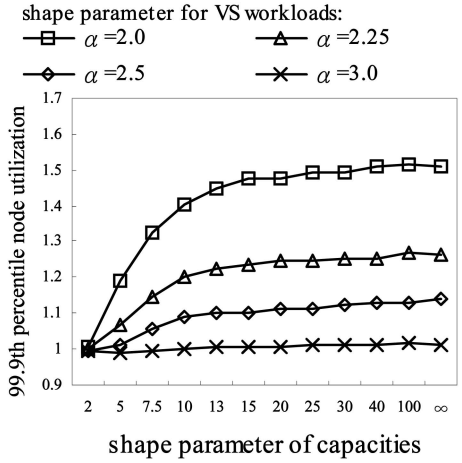

Fig. 8. DSLS 99.9th percentile node utilization performance for all problem instances.

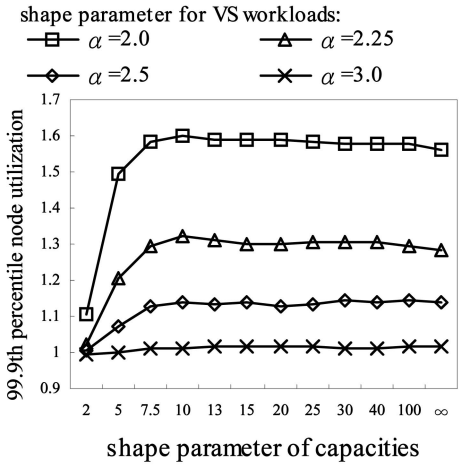

Fig. 9. M2M 99.9th percentile node utilization performance for all problem instances.

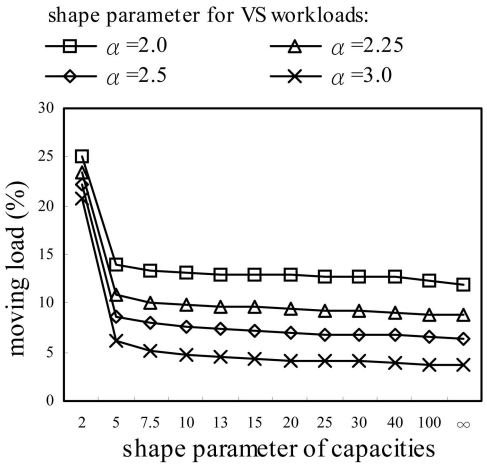

Fig. 10. Moving load ratio versus the shape of capacities for all problem instances.

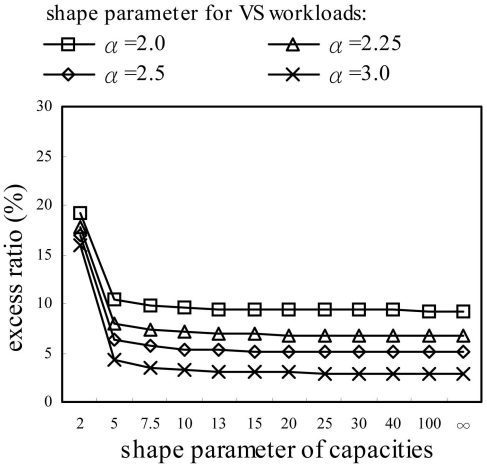

Fig. 11. Excess workload ratio over node capacities for all problem instances.

considered. Figs. 8 and 9 depict the node utilization result for DSLS and M2M.

When the workloads are relatively homogeneous ($\alpha_w \geq 3$), the 99.9th percentile utilization of both algorithms stays close to or under 1, implying that feasible solutions are found for most of the instances. When workloads become heterogeneous, the 99.9th percentile utilizations of both algorithms are more than 1. DSLS does a better job in moving the VSs so that the 99.9th percentile utilization is smaller across the whole spectrum of heterogeneity, especially when capacities are more homogeneous.

4.4 Moving Load Performance

Fig. 10 depicts the total load moved fraction result for DSLS for all instances, whereas the ratio of the total excess workload to the total system load is shown in Fig. 11. The total excess workload is the total amount of workload exceeding the node capacity for all nodes in the system. Fig. 10 shows that the total load moved fraction decreases quickly as the node capacity becomes more homogeneous. This is because, when the node capacity is highly heterogeneous, there are many small-capacity nodes, which are highly likely to be overloaded and need to have their VSs moved. Figs. 10 and 11 in combination demonstrate that DSLS moves very little more than what must be moved and is therefore a very effective algorithm.

4.5 Performance at Varying Levels of System Utilization in a Static Environment

The success ratio performance at various levels of system utilization is evaluated for admissible instances. System utilization is the ratio of the mean of the total VS workloads relative to the mean of the total node capacities. The system utilization examined varies from 0.5 to 0.95. We only examine the performance of the algorithms under the most difficult scenario. The results are shown in Fig. 12. Not surprisingly, as the system utilization increases, the success ratio of the algorithms decreases, since increasing system utilization results in harder-to-solve problem instances. It is clear that DSLS holds a significant advantage over other algorithms.

4.6 Performance under Varying Numbers of VSs

In real environments, the number of VSs on each node may change in response to changes in the workload. We have observed from the dynamics of M2M that the number of VSs on each node is generally proportional to its capacity,

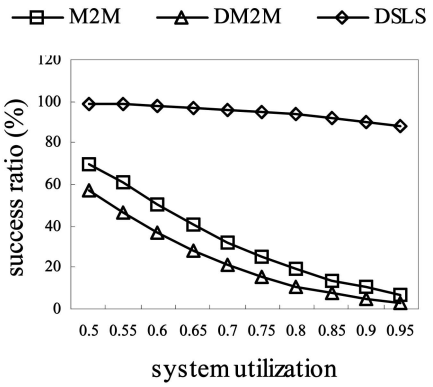

Fig. 12. Success ratio versus utilization for admissible instances.

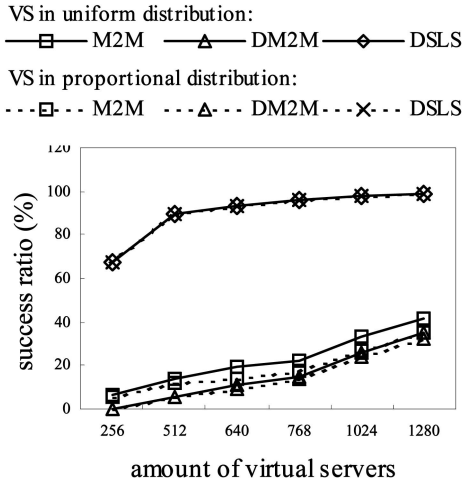

Fig. 13. The success ratios versus varying numbers of VSs in a uniform and a proportional distribution.

whereas, in k-choices [9], the number of VSs on each node is relatively constant. To understand the impact, the performance of the algorithms is evaluated under two types of association of the number of VSs to nodes: uniform and proportional. In uniform association, every node is assigned the same number of VSs, whereas, in proportional association, the number of VSs assigned to a node is proportional to its capacity. The algorithms are then evaluated under the most difficult scenario. The total number of VSs is varied from two to 10 times the number of nodes in the system. In Fig. 13, we plot the success ratio performance for admissible instances because it appropriately characterizes the impact of varying the number of VSs.

Fig. 13 shows that the success ratios increase as the number of VSs per node increases, under both uniform and proportional VS assignments. This is because the average VS size decreases as the number of VSs increases, allowing more possibilities for VS swapping for load balance actions. The different association schemes do not have a large impact for each algorithm. In addition, as long as a sufficient number of VSs is employed, for example, four VSs per node on the average, increasing the number of VSs has a minimal impact on the success ratio as well. These observations are consistent with the findings in previous studies [9].

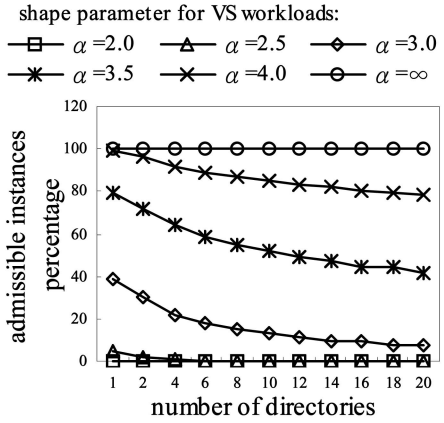

Fig. 14. The admissibility property under the random registration policy.

4.7 Performance in a Multidirectory Static Environment

More than one directory node must be employed in order to lessen the burden on a single directory and speed up the load balancing process [6]. Two fundamental questions must be addressed with regard to the directory mechanism. First, there is a trade-off in the number of directory nodes employed. Employing more directory nodes reduces the computation overhead per directory, but the number cannot be too many to make the search space for possible exchange of VSs too limited. The second question is the manner for nodes to register with directory nodes. This issue is important because, for example, when nodes exhibit high capacity heterogeneity, there are only few nodes with very large capacities. If the few large-capacity nodes all register with the same directory node, there is no way to balance the system load. The following registration policies are considered in this paper:

- *Random registration policy.* A physical node selects a random directory node with which to register. This is the policy employed in the M2M scheme.
- *Uniform-share registration policy.* The share value of a node is defined to be the ratio of the total workload relative to the capacity of the node. Under this policy, the nodes are *assigned* to directory nodes in such a way that the total share value registered with each directory node is approximately equal. We first sort the share value of all physical nodes in decreasing order and then assign the physical nodes to the directories by starting from the one with the largest share value and then in the reverse order when the last one is reached.

In the following experiments, we only examine the performance when node capacity $\alpha_c = 10$, as in the most difficult scenario. The number of directories is varied from one to 20. The total number of nodes is 1,280, so that when the number of directory nodes is 10, the results from previous experiments can be compared. We first characterize the admissibility property in this setting. Figs. 14 and 15 depict the percentage of admissible problem instances when nodes register under the random and uniform-share policies, respectively.

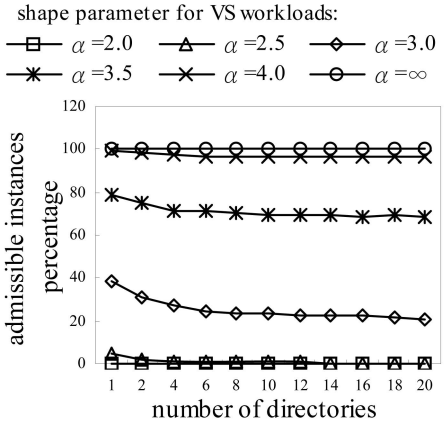

Fig. 15. The admissibility property under the uniform-share registration policy.

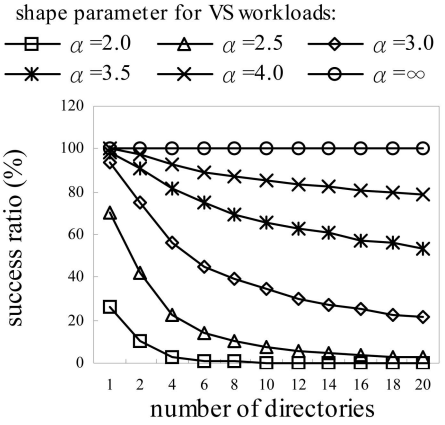

Fig. 16. Network-level success ratio performance for admissible instances under the random registration policy.

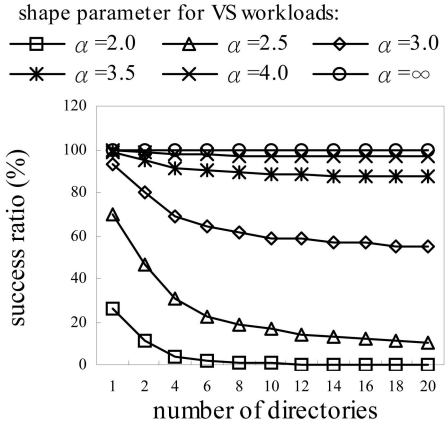

Fig. 17. Network-level success ratios for admissible instances under the uniform-share registration policy.

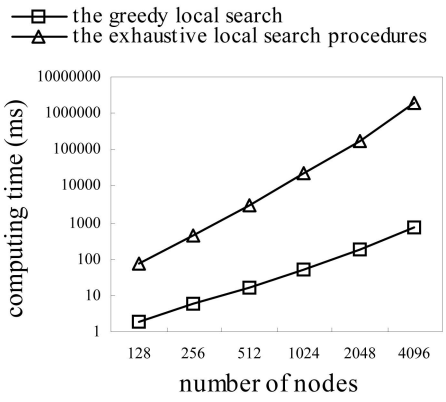

Fig. 18. DSLS computing time (in milliseconds) under varying numbers of nodes.

It can be seen that employing multiple-directory nodes have a dramatic impact on the admissibility property. As the node capacities are relatively homogeneous in this most difficult scenario, any amount of heterogeneity in the workload, $\alpha_w < 4$, decreases the admissible percentage significantly as the number of directories increases. The uniform-share policy can assign a VS to nodes so they match better, yet beyond $\alpha_w = 4$, even the uniform-share policy cannot improve the admissibility situation. For example, when $\alpha_w = 2.5$, comparing Figs. 3 and 14, almost 51 percent of the instances are admissible in the single-directory case, whereas almost none of the instances are admissible in the multiple-directory case.

We have evaluated the success ratio performance of DSLS for all instances when nodes register under the random and uniform-share policies. The results look quite similar to Figs. 14 and 15. Therefore, we present its performance for admissible instances instead. Figs. 16 and 17 plot DSLS's performance for admissible instances under random and uniform-share policies, respectively. It achieves better performance under the uniform-share registration policy than under random policy. The performance gap may be quite significant. For example, when the workload shape value is 4.0 and the number of directory nodes is 20, DSLS can achieve a near-100-percent success ratio under the uniform-share policy yet can only achieve a 78 percent success ratio under the random policy.

In Fig. 6, the success ratio is close to 100 percent under a single directory when $\alpha_w \geq 3$ and $\alpha_c = 10$. However, in Fig. 17, the success ratio is close to 58 percent when $\alpha_w = 3$. The reason is twofold. First, Fig. 17 depicts network-level success ratios, which means feasible solutions must be found for all directory nodes. Chances for network-level success are significantly smaller. Second, the admissible conditions are checked with respect to problem instances. It may happen that an admissible instance may not be feasible at a directory level after VSs are assigned to directory nodes. In contrast to previous studies that concluded that the number of directories employed has no influence on algorithm performance, our results show that the number of directories employed has a significant impact.

4.8 Execution Time in a Static Environment

The scalability property of DSLS under the greedy and the exhaustive local search procedures is evaluated in terms of their execution time. On a Pentium 4 2.93-GHz machine, the execution time performance is shown in Fig. 18.

The greedy local search outperforms the generic exhaustive local search by more than three orders of magnitude for larger network configurations. DSLS using a greedy local search takes less than 1 second to execute for

TABLE 2
Parameters under Capacity and Load Heterogeneity in a Dynamic Environment

Parameters	Value range
Object arrival process	Poisson, mean inter-arrival time of 0.01 second
Number of objects	~ 100,000
Object lifetime	Exponential, calculated to reach number of objects
Object workload	Pareto $\alpha_o = 2.0 \sim \infty$, β_o=3.125 ~ 6.25 , determined by utilization
Number of nodes	1280
Number of directory nodes	10
Node registration policy	Uniform share
Number of VSs per node	5
Location of VSs	Uniformly distributed in ID-space
Node capacity	Pareto $\alpha_c = 2.0 \sim \infty$, β_c=3.125 ~ 6.25
System utilization	0.8
Load balance period	60 seconds

a directory containing 4,096 nodes. It is practical for systems that require relatively frequent load balancing.

4.9 Performance in a Dynamic Environment

The algorithms are also evaluated in a dynamic storage-oriented environment similar to the one in [7], in which objects arrive at and leave the system according to the Poisson process, but physical nodes do not. The parameters used in the experiments are shown in Table 2. Note that the object shape parameter is denoted by symbol α_o. As the object size distribution is Pareto, the workload of a VS is the sum of Pareto random variables. Nonexplicit expressions for such sums can be found in [5]. It suffices for our purpose to know that the distribution of sums of Pareto variables behaves like a Pareto distribution *in the tail*.

We first characterize the admissibility region in a dynamic environment in terms of the percentage of time instants when the problem instances are found to satisfy the necessary conditions among all instants. The examined time instants are chosen to be right before each scheduled load balance action, after nodes have registered with the directory nodes. Note that the results are dependent on the specific registration policy employed.

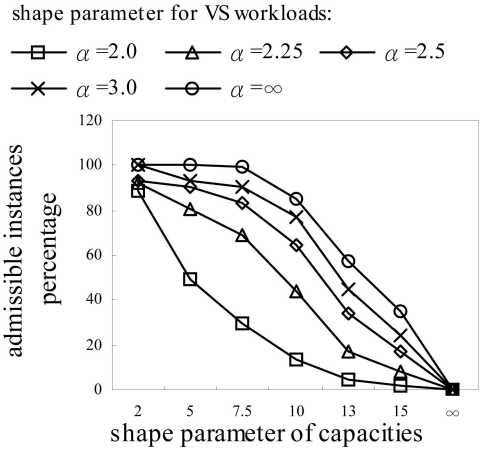

Fig. 19. The admissible instance percentage in a dynamic environment.

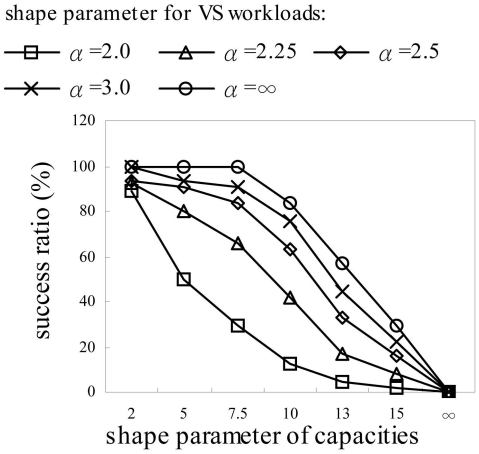

Fig. 20. DSLS network-level success ratio performance for all instances under the uniform-share policy.

The results, shown in Fig. 19, look similar to those in the static case. However, at more homogeneous capacity shape values, $\alpha_c \geq 10$, the admissible percentages are smaller than those in Fig. 3. For $\alpha_c \geq 15$, there are almost no admissible instances at all. This result is because of the $\Theta(\log N)$ imbalance in VS zone sizes. Using the expression in [3], $f_{\max}$ is found to be between 7 and 17 with a probability of 0.999. Therefore, the largest VS holds many more objects than average ones, resulting in lower admissible percentages when $\alpha_c \geq 10$. Again, in [6] and [7], the particular capacity shape value chosen is 2, so the easier scenarios were studied.

Fig. 20 depicts the success ratio result for all instances. Again, it is more insightful to examine the success ratio for admissible instances, which is plotted in Fig. 21.

Fig. 21 shows that the dynamic case is consistent with the multidirectory static case, exhibited in Fig. 17, in that higher

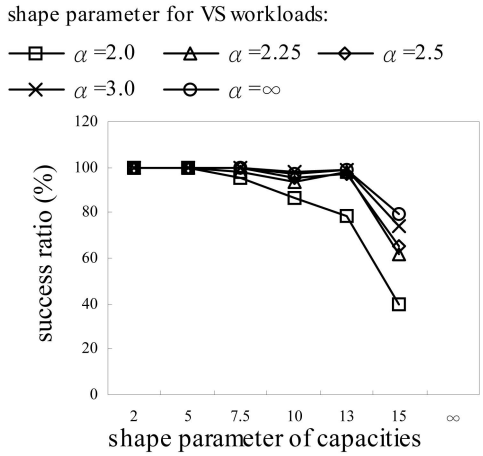

Fig. 21. DSLS network-level success ratio performance for admissible instances under the uniform-share policy.

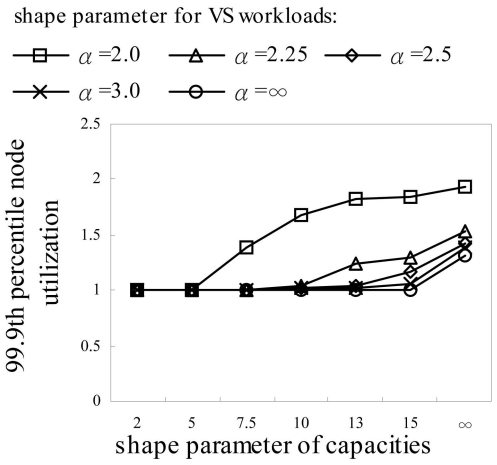

Fig. 22. DSLS network-level 99.9th percentile node utilization for all problem instances under the uniform-share policy.

success ratios occur when capacities are more heterogeneous and workloads are more homogeneous. However, for a wide range of node capacity shape values, $\alpha_c \leq 10$, the success ratios are very high. Examining the variance in VS sizes reveals that this is a direct result of the fact that the variances are quite small, producing relatively easy problem instances. Beyond $\alpha_c \geq 13$, the success ratios are still higher than those when the number of directory nodes is 10 in Fig. 17, again for the same reason. Similar observations can be made with respect to the results for the 99.9th percentile node utilization, which is plotted in Fig. 22.

5 Conclusions and Future Work

This paper studied the VS framework for solving the load balance problem in a structured P2P system. The first main contribution is an effective and efficient DSLS algorithm, which leverages work in GAP. The second contribution is an in-depth analysis of the effect of capacity and workload heterogeneity on algorithm performance in both static and dynamic environments and the qualitative relationship between static and dynamic environments. We plan to investigate the following important issues in the future: We intend to explore other cost-reducing neighborhoods to further improve the DSLS algorithm. As the variance of a VS workload has a significant impact on the success ratio performance, we plan to investigate VS merging and splitting strategies to enhance the performance of the algorithms. We also plan to investigate distributed protocols to implement the proposed node registration policies. Previous work in the parallel balanced allocation problem [28], [29] seems to be closely related to our problem. We also plan to perform a more in-depth study of issues in the dynamic scenario in which a node joins and leaves the system.

References

[1] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM SIGCOMM '01,* pp. 149-160, 2001.

[2] F. Dabek, M. Kaashoek, D. Karger, D. Morris, and I. Stoica, "Wide-Area Cooperative Storage with CFS," *Proc. 18th ACM Symp. Operating Systems Principles (SOSP '01),* pp. 202-215, Oct. 2001.

[3] X. Wang and D. Loguinov, "Load-Balancing Performance of Consistent Hashing: Asymptotic Analysis of Random Node Join," *IEEE/ACM Trans. Networking,* vol. 15, no. 5, Oct. 2007.

[4] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," *Proc. 29th Ann. ACM Symp. Theory of Computing (STOC '97),* pp. 654-663, 1997.

[5] C. Ramsay, "The Distribution of Sums of Certain I.I.D. Pareto Variates," *Comm. in Statistics—Theory and Methods,* vol. 35, pp. 395-405, 2006.

[6] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Structured P2P Systems," *Proc. Second Int'l Workshop Peer-to-Peer Systems (IPTPS '03),* Feb. 2003.

[7] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Dynamic Structured P2P Systems," *Proc. IEEE INFOCOM,* 2004.

[8] P.B. Godfrey and I. Stoica, "Heterogeneity and Load Balance in Distributed Hash Tables," *Proc. IEEE INFOCOM,* 2005.

[9] J. Ledlie and M. Seltzer, "Distributed, Secure Load Balancing with Skew, Heterogeneity, and Churn," *Proc. IEEE INFOCOM,* 2005.

[10] X. Wang, Y. Zhang, X. Li, and D. Loguinov, "On Zone-Balancing of Peer-to-Peer Networks: Analysis of Random Node Join," *Proc. ACM SIGMETRICS '04,* pp. 211-222, June 2004.

[11] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M.F. Kaashoek, J. Kubiatowicz, and R. Morris, "Efficient Replica Maintenance for Distributed Storage Systems," *Proc. Third Symp. Networked System Design and Implementation (NSDI '06),* May 2006.

[12] H. Weatherspoon, "Design and Evaluation of Distributed Wide-Area On-Line Archival Storage Systems," PhD dissertation, Electrical Eng. and Computer Sciences (EECS) Dept., Univ. of California, Berkeley, Oct. 2006.

[13] M. Laguna, J.P. Kelly, J.L. Gonzalez Velarde, and F. Glover, "Tabu Search for the Multilevel Generalized Assignment Problem," *European J. Operational Research,* vol. 82, pp. 176-189, 1995.

[14] M.L. Fisher, R. Jaikumar, and L.N. Van Wassenhove, "A Multiplier Adjustment Method for the Generalized Assignment Problem," *Management Science,* vol. 32, no. 9, pp. 1095-1103, 1986.

[15] G.T. Ross and R.M. Soland, "A Branch and Bound Algorithm for the Generalized Assignment Problem," *Math. Programming,* vol. 8, no. 1, pp. 91-103, 1975.

[16] M.G. Narciso and L.A.N. Lorena, "Lagrangean/Surrogate Relaxations for the Generalized Assignment Problems," *European J. Operational Research,* vol. 114, no. 1, pp. 165-177, 1999.

[17] J.A. Diaz and E. Fernandez, "A Tabu Search Heuristic for the Generalized Assignment Problem," *European J. Operational Research,* vol. 132, pp. 22-38, 2001.

[18] P.C.H. Chu and J.E. Beasley, "A Genetic Algorithm for the Generalised Assignment Problem," *Computers and Operations Research,* vol. 24, pp. 17-23, 1997.

[19] M. Yagiura, T. Yamaguchi, and T. Ibaraki, "A Variable Depth Search Algorithm for the Generalized Assignment Problem," *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization,* S. Vosß, S. Martello, I.H. Osman, and C. Roucairol, eds., Kluwer Academic Publishers, pp. 459-471, 1999.

[20] H.R. Lourenco and D. Serra, "Adaptive Search Heuristics for the Generalized Assignment Problem," *Mathware and Soft Computing,* vol. 9, pp. 209-234, 2002.

[21] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology," *Proc. ACM SIGCOMM '99,* pp. 251-262, 1999.

[22] L.A. Adamic, *Zipf, Power-Laws, and Pareto—A Ranking Tutorial,* http://www.hpl.hp.com/research/idl/papers/ranking/ranking.html, 2000.

[23] S. Sahni and T. Gonzalez, "P-Complete Approximation Problems," *J. ACM,* vol. 23, no. 3, pp. 555-565, July 1976.

[24] T. Stutzle and H. Hoos, "The Max-Min Ant System and Local Search for Combinatorial Optimization Problems," *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization,* S. Vosß, S. Martello, I.H. Osman, and C. Roucairol, eds., Kluwer Academic Publishers, pp. 313-329, 1999.

[25] M.G.C. Resende and C.C. Ribeiro, "Greedy Randomized Adaptive Search Procedures," *State-of-the-Art Handbook of Metaheuristics*, F. Glover and G. Kochenberger, eds., Kluwer Academic Publishers, 2002.
[26] M. Dorigo and T. Stützle, *Ant Colony Optimization.* MIT Press, 2004.
[27] E.W. Weisstein, "Pareto Distribution," *MathWorld—A Wolfram Web Resource,* http://mathworld.wolfram.com/ParetoDistribution.html, 2007.
[28] V. Stemann, "Parallel Balanced Allocations," *Proc. Eighth Ann. ACM Symp. Parallel Algorithms and Architectures (SPAA '96)*, pp. 261-269, 1996.
[29] P. Berenbrink, F. Meyer auf der Heide, and K. Schroder, "Allocating Weighted Jobs in Parallel," *Theory of Computing Systems*, vol. 32, no. 3, pp. 281-300, 1999.

Chyouhwa Chen received the PhD degree from the State University of New York, Stony Brook. He is an associate professor in the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taiwan. His research interests include computer network, traffic management, multimedia communications, and issues in multicast routing.

Kun-Cheng Tsai received the MS degree in electrical engineering from Chung Hua College, Taiwan, in 1995 and the PhD degree in electrical engineering from the National Taiwan University of Science and Technology in 2005. During 1997-2002, he stayed in the Multimedia and Communications Research Laboratory, National Taiwan University of Science and Technology. Currently, he is a senior engineer and one of the lead developers of sensor networks at the Institute for Information Industry, Taiwan. He is now studying real-time multimedia communication and wireless sensor networks.