scc

Informed Provisioning of Storage for Cluster Applications

HARSHA V. MADHYASTHA, JOHN C. MCCULLOUGH, GEORGE PORTER, RISHI KAPOOR, STEFAN SAVAGE, ALEX C. SNOEREN, AND AMIN VAHDAT

Harsha V. Madhyastha is an Assistant Professor in Computer Science and Engineering at the University of California, Riverside. His research interests include distributed systems, networking, and security.
harsha@cs.ucr.edu

John C. McCullough is pursuing a PhD in computer science at the University of California, San Diego, and is advised by Professor Alex C. Snoeren. He received an MS in computer science from UCSD (2008) and a BS in computer science from Harvey Mudd College.
jmccullo@cs.ucsd.edu

George Porter is a Research Scientist in the Center for Networked Systems and a member of the Systems and Networking Group at UC San Diego. He received his BS from the University of Texas at Austin and his PhD from the University of California, Berkeley.
gmporter@cs.ucsd.edu

Rishi Kapoor is a PhD student in computer science at the University of California, San Diego. His research interests include cloud computing and computer systems.
rkapoor@cs.ucsd.edu

Stefan Savage is a Professor of Computer Science at the University of California, San Diego. He has a BS in history and reminds his colleagues of this fact anytime the technical issues get too complicated.
savage@cs.ucsd.edu

Alex C. Snoeren is an Associate Professor in the Computer Science and Engineering Department at the University of California, San Diego, where he is a member of the Systems and Networking Research Group. His research interests include operating systems, distributed computing, and mobile and wide-area networking.
snoeren@cs.ucsd.edu

Amin Vahdat is a Principal Engineer at Google and also the SAIC Professor in the Department of Computer Science and Engineering at the University of California, San Diego. Vahdat's research focuses broadly on computer systems, including distributed systems, networks, and operating systems.
vahdat@cs.ucsd.edu

Today, application providers can choose from a range of storage choices to provision their infrastructure for cluster-based applications. Each storage technology presents a different point in a complex tradeoff space of cost, capacity, and performance. To help application providers choose from these alternatives, we developed scc [1] to automate the selection of cluster storage configurations based on a formal specification of applications, hardware, and workloads. Our tool allows administrators to understand how high-level workload characteristics influence the cluster architecture, and in applying scc to several representative deployment scenarios, we show how it can enable 2x–4.5x cost savings when compared to traditional scale-out techniques.

Identifying an appropriate cluster architecture to host a large-scale service is often not straightforward. Given a set of resources to choose from (e.g., as shown in Table 1), an application provider has to answer several questions. What storage technologies should be employed, and how should data be partitioned across them? Where should caching be employed? What types of servers should be chosen to house the selected storage units?

In addition, even if the application's implementation is efficient and there is coarse-grained parallelism in the underlying workload, how will algorithmic shifts in the application or variations in workload affect the appropriate cluster architecture? Our goal is to automate the process of answering these questions, rather than relying solely on human judgment.

Resource	MB/s	IOPS	Watts	Cost
7.2K Disk (500 GB)	90 (R) 90 (W)	125 (R) 125 (W)	5	$213
15K Disk (146 GB)	150 (R) 150 (W)	285 (R) 185 (W)	2.3	$296
SSD (32 GB)	250 (R) 80 (W)	2500 (R) 1000 (W)	2.4	$456
DRAM (1 GB)	12.8K (R) 12.8K (W)	1.6B (R) 1.6B (W)	3.5	$35
CPU core	—	—	20	$137

Server type	Resource Limits	Cost
Server1	4 cores, 1 Gbps network 12GB DRAM, 4 SAS slots	$1400
Server2	16 cores, 10 Gbps network 48GB DRAM, 16 SAS slots	$1850
Server3	32 cores, 10 Gbps network 512GB DRAM, 16 SAS slots	$11000

Table 1: Example set of hardware units input to scc. Cost is price plus energy costs for three years.

In developing scc, we show how to systematically exploit storage diversity, i.e., select among different physical media, local and remote storage, and various caching strategies. First, we determine how the characteristics of applications, workloads, and hardware should be specified in order to automate the selection of cluster configurations. To do so, we study several representative deployment scenarios and identify a parsimonious yet sufficiently expressive set of parameters that capture the tradeoffs offered by different types of storage devices and the varying demands across application components. To characterize applications, we leverage developer knowledge and standard techniques to trace the execution of applications, and, once developed, application models can be reused across deployments. Second, we implement scc, a storage configuration compiler, to take specifications of applications, workloads, and hardware as input, automatically navigate the large space of storage configurations, and zero in on the configuration that meets application SLAs at minimum cost.

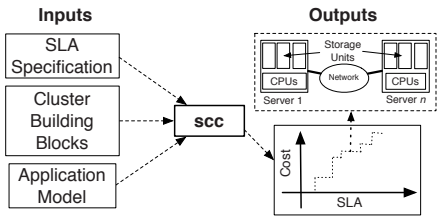


Figure 1: scc takes formal specifications of applications, hardware, and SLA metrics as input. It outputs a cost-versus-SLA distribution, while determining the minimum cost cluster configuration for every SLA value.

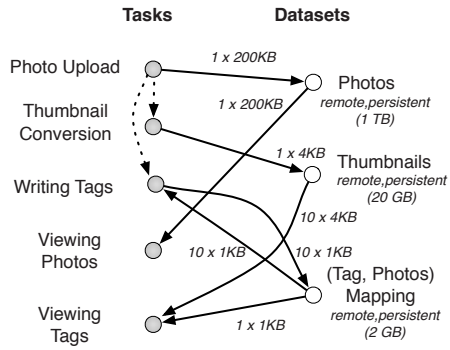


Figure 2: Interaction between tasks and datasets in example photo-sharing application. Edges between tasks and datasets represent I/O with direction differentiating input and output. Dotted edges indicate task dependencies.

Specifying scc's Inputs

As shown in Figure 1, scc takes three inputs: (1) a model of application behavior, specified in part by the application's developer and in part by the administrator deploying the application; (2) characteristics of available hardware building blocks specified by the infrastructure provider; and (3) application performance metrics, i.e., a parameterized service level agreement (SLA) (e.g., a Web service SLA might specify a peak query rate per second). Given these inputs, scc computes how cluster cost varies as a function of the SLA and outputs a low-cost cluster configuration that meets the SLA at each point in the space. scc's output cost vs. SLA value distribution helps administrators decide what performance can be supported cost effectively.

While there has been prior work on similarly configuring storage based on formal specifications of workloads and hardware [2, 3], these prior approaches take as input the workload demands on every component of the application (e.g., the I/O rates to be satisfied by a logical volume of data). In practice, application providers seek to satisfy SLAs that are specified at a higher level. For example, in a photo-sharing Web service, the target may be to cope with a certain rate of photo uploads and downloads. To translate such SLA requirements into demands on individual application components, we need a model of the application.

Our characterization of applications accounts for two aspects: its implementation and the workload in its planned deployment. To capture an application's implementation, we first ask the application's developer to describe its decomposition into compute and storage components, and the interaction between them. We account for various characteristics of these components, such as whether the application runs in multiple phases, the I/O operations it performs in response to particular inputs, and the dependencies between different parts of the application.

For example, Figure 2 depicts the components, and the interaction between them, for an example photo-sharing Web service. Although we place the onus on application developers to formally specify the components of their application, an application's specification is reusable across deployments.

Second, we enable those who deploy an application to annotate the specification of the application's architecture with properties of the expected workload in their deployment. To do so, we require that the compute and I/O characteristics of an application's components, when subjected to the target workload, be determined by running small-scale application benchmarks. We characterize compute components by their memory requirements and storage components by their storage capacity and persistence needs. We also label I/O operations and inter-component dependencies with properties such as the record size being read/written, and whether these operations are synchronous or asynchronous. The former helps differentiate between random and sequential I/O, while the latter determines the application's ability to trade off latency with throughput. Extracting these properties requires tracing the application's execution, now standard practice in resource-intensive performance-critical applications. In the absence of built-in tracing support, systems like Magpie [4] can be leveraged.

Automating the Navigation of the Configuration Space

scc determines the cost versus SLA distribution for a given application deployment by considering the configuration for each point in the distribution independently.

To compute the cluster configuration for a target SLA, scc needs to determine the *architecture* of the cluster (the types of storage media to be used for each dataset and the types of servers used to host storage units and CPUs) and the *scale* at which this architecture must be instantiated (the number of servers, storage units, and CPUs, as well as the level of parallelism of each application task) to meet the SLA.

Guiding Principles

Two key principles help scc identify the right cluster configuration. First, the architecture and scale for every application component can be determined independently when all operations are performed asynchronously, but not when some operations are synchronous. The SLA for any task only specifies the rate at which a task's execution path must run. In the typical case, where a task's execution path contains some operations that block others, scc needs to determine the "division of labor" across these operations that minimizes cost. For example, in a task that reads from an input dataset and then writes to an output dataset, in order to meet the task's SLA it may suffice to provision fast storage for any one of the two datasets; provisioning fast storage for both datasets may unnecessarily result in higher cost due to storage capacity requirements, whereas slow storage for both may incur higher costs in satisfying I/O throughput needs. Hence, scc jointly determines resource requirements across all application components.

Second, since scc provisions for peak load, it prevents over-provisioning by ensuring that at least one resource is bottlenecked on every server at peak load. (If the application provider wants to run the cluster at lower peak utilization, that can be specified as input.) Based on our characterization of hardware, there are four possible bottlenecks on each server: (1) the number of slots, (2) the bandwidth on an I/O controller, (3) the number of CPU cores, (4) network bandwidth.

Algorithm

Driven by the need for joint optimization across components, scc represents each point in the configuration space by the assignment of storage unit types to datasets. This assignment suffices to represent each configuration because, given this information, we can compute the number of storage units of each type and the number of CPUs necessary to meet the SLA. We can then compute the number of servers of each type required to accommodate these resources. As a result, if S is the number of storage choices and D is the number of datasets, scc has to search through a space of $O(S^D)$ configurations; for each dataset, scc can choose any one of the S storage options.

In cases where the configuration space is too large to perform an exhaustive search, scc performs a repeated gradient descent search. We start with a randomly chosen configuration.

In each step, we consider all neighboring configurations—those which differ in exactly one dataset's storage-type assignment—and move to the configuration that still meets the SLA with the maximum decrease in cost. We repeat this step until we find a configuration where all neighbors have higher cost. Since gradient descent can lead to a local minimum, we repeat this procedure multiple times with different randomly chosen initial configurations and settle on the minimum cost output across the multiple attempts. In our evaluation, we have found that repeating the gradient descent 10 times is typically sufficient to find a solution close

to the global minimum. Therefore, even when determining the configuration to satisfy workloads of tens of thousands of queries per second, scc's running time for any particular SLA is within a minute.

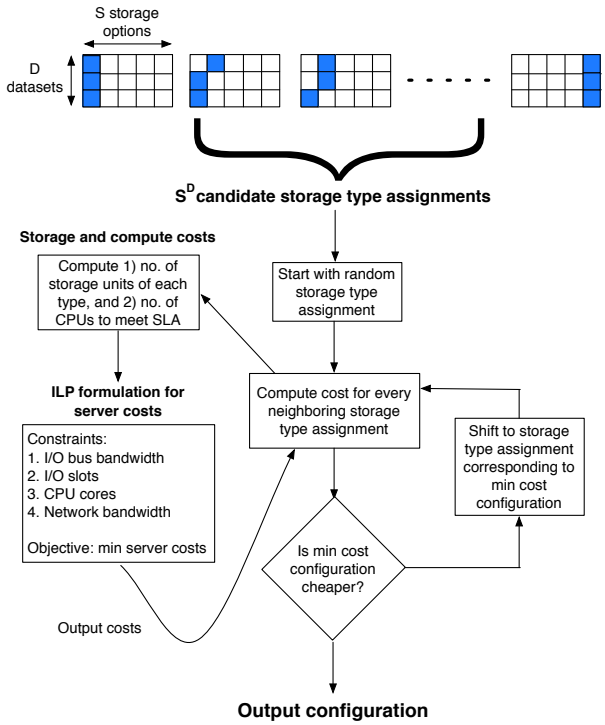


Figure 3: scc represents every configuration by the storage type assignments for each of the application's datasets, and searches through this space with gradient descent (with multiple randomly chosen initial configurations) to find the minimum cost configuration.

At the heart of scc's search of the configuration space (summarized in Figure 3) is a procedure that, given any particular assignment of storage types to datasets, determines a cost-effective set of resources to meet the target SLAs. In this procedure, scc first determines for each remote dataset (i.e., not local to any task) the number of storage units required of the type assigned to the dataset in the configuration state. Second, scc determines the number of CPUs required by every task and the number of storage units of the assigned type needed by the task's local datasets. Finally, it solves a linear integer program to determine the types of servers and number of each kind required to minimize overall cluster cost.

Heterogeneous Configurations Beat Scale-Out

We have applied scc to three distributed applications with distinctly different workload characteristics: (1) a product search Web service modeled on Google Merchant Center, (2) Terasort, a MapReduce job to sort large tuple collections, and (3) a photo-sharing Web service modeled on Flickr. We validated scc by deploying these applications on a range of cluster configurations and measuring application performance on these configurations.

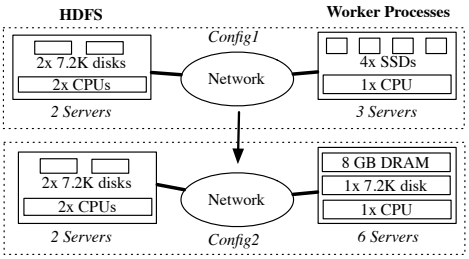
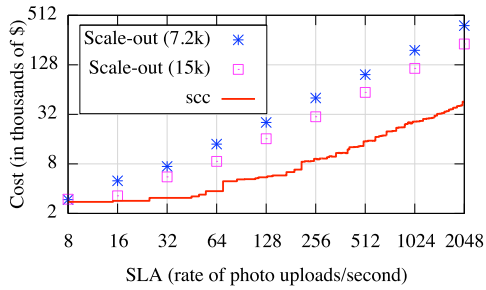


Figure 4: Illustration of transition in minimum cost cluster configuration recommended by scc, when input workload is increased. scc uses heterogeneous architectures to reduce costs in comparison to simply scaling out resources.

In applying scc to these diverse application workloads, we repeatedly find that clusters with heterogeneity—rather than conventional homogeneity—across servers are necessary to optimize cost. The resources required differ across application components due to the varying ratios of capacity, compute, and I/O throughput needs across components. Figure 4 shows an example of how scc's recommended configuration for our example product search Web service changes when the input workload is increased. First, we note that different application components are hosted on servers equipped with different types of storage. Second, the types of hardware resources allocated to the same application component radically change, rather than resources simply being increased in quantity, when the workload is increased.

Transitions in Cost-Optimal Storage Configurations

In applying scc to our exemplar applications, we also find that the most cost-effective cluster architecture depends not only on the application being provisioned but also on the workload and performance requirements. Data that was initially capacity-bound may become I/O-bound at higher loads, calling for shifts from high capacity but slow storage, e.g., disks, to low capacity but fast storage, e.g., SSDs. As a result, cluster configurations output by scc for our exemplar photo-sharing and product search applications result in 2x–4.5x average savings in cost compared to similarly performant scale-out options.



(a)

Uploads/s	Storage unit type		
	Photos	*Thumbnails*	*Tags*
≤ 5	Disk	Disk	Disk
5–25	Disk	Disk	Disk + DRAM
25–330	Disk	SSD	Disk + DRAM
330–930	SSD	Disk + DRAM	Disk + DRAM
930–10k	Disk + DRAM	Disk + DRAM	Disk + DRAM

(b)

Figure 5: (a) Cost versus SLA distribution output by scc for example photo-sharing application, with (b) the corresponding regimes in the cost-effective architecture. Simply scaling out alternate configurations inflates cost by 3x–4.5x on average.

As an example, Figure 5(a) shows the cost distribution output by scc across a range of SLA values for our photo-sharing application. Perhaps surprisingly, no huge spikes are observed in this distribution; this is because scc balances costs across the kind of storage, the number of CPUs, and the number of machines provisioned. Rather than adding more machines of the same type, the cluster architecture transitions to faster storage as the SLA becomes more stringent, with transitions in storage type for different datasets seen at different SLA values.

Figure 5(b) highlights these transitions. Note that the quantity in which different types of resources are provisioned varies within each architecture regime specified by every row in the table.

We further compare the cost output by scc with the cost associated with a scale-out approach. We compare the scc configuration to the cases where the building block is based around: (1) storage servers with four 7.2k-RPM disks (the cost-optimal storage type for all datasets at the lowest SLA), and (2) servers with four 15k-RPM disks. In either case, more storage servers are added as the required rates increase. Figure 5(a) shows that the costs in both cases are significantly greater than with scc, incurring between 3 and 4.5 times more cost (note the logarithmic y-axis). Thus, simply scaling out a homogeneous configuration that is cost-effective at low loads can result in significant cost inflation at higher loads.

How Robust Are scc's Recommendations?

scc's output cluster configuration for a target SLA is a function of both the SLA and the values specified for the various attributes in the application and hardware specifications. In practice, an administrator may not have precise values for all attributes due to incomplete knowledge of the application workload, uncertainty of hardware costs, or measurement inaccuracy in benchmarking the application.

Attribute	**Range with same architecture**		
	Lowest value	Input value	Highest value
Avg. photo size	50 KB	200 KB	850 KB
Avg. thumbnail size	1 KB	4 KB	30 KB
SSD unit price	$200	$450	$900

Dataset	**Most sensitive to what change in hardware costs?**
Photos	20% drop in $ of 7.2K-RPM disk
Thumbnails	92% drop in $ of DRAM
Tags	31% drop in $ of 15K-RPM disk

Table 2: (a) Robustness of scc's output with respect to input values for a sample set of attributes; (b) the change in hardware costs to which scc's storage decision for each dataset is most sensitive.

scc is naturally built to cope with such uncertainty. For every attribute in the input specifications, scc varies the value of the attribute in the neighborhood of the initially specified value. For each attribute, it then outputs the range of values for that attribute wherein the cost-effective cluster architecture, i.e., the types of resources

assigned to different application components, remains unchanged; variance of the attribute's value within this range can be handled by simply adding more resources of the same type. Outside of that range, the cluster will need to be revamped with a different type of resource for some application component, a more onerous undertaking. For example, we consider our example photo-sharing service with an SLA of 100 uploads/s, 300 photo views/s, and 100 tag views/s. Table 2(a) shows the value ranges output by scc for a few attributes, within which the cluster architecture is robust to change. For example, we see that as long as the average photo size remains between 50 KB and 850 KB, the cluster architecture remains the same as that obtained with the input value of 200 KB.

Furthermore, scc can also evaluate the sensitivity of its choice of storage configuration for every dataset in the application. For example, consider our photo-sharing Web service again with the same input SLA as above. Based on current hardware costs, scc determines that photos be stored on 15k-RPM disks, thumbnails be stored on SSDs, and tags be stored persistently on 7.2k-RPM disks and cached in DRAM, in order to meet the SLA at minimum cost. However, these recommendations are likely to change as prices for storage units drop. scc can determine the robustness of its storage option choice in response to such changes in hardware prices. To do so, it varies the price of every type of storage unit from its input value down to 0, and notes the inflection points at which the optimal storage choice for some dataset changes. Based on this analysis, it can determine, for every dataset, that change in hardware price to which the current storage choice for the dataset is most sensitive. Table 2(b) shows that while the storage choices for photos and tags are sensitive to relatively small reductions in the prices for 7.2k-RPM and 15k-RPM disks, scc's recommendation of storing thumbnails on SSDs is very robust to price fluctuations.

Conclusion

The primary thesis of our work is that the choice of cluster hardware for an application should be informed by the interaction between the application's behavior and the properties of hardware. Rather than relying on human judgment to do so, we developed scc to compile formal specifications of these inputs into cost-effective cluster configurations. We have applied scc to a range of application workloads and storage options to demonstrate that scc captures sufficient detail to identify the appropriate hardware at any given scale. We find that scc often recommends heterogeneous cluster architectures that result in significant cost savings compared to traditional scale-out approaches.

Our implementation of scc is available for download at http://www.cs.ucr.edu/~harsha/scc/.

Acknowledgments

This work was supported in part by a NetApp Faculty Fellowship and through a grant from the National Science Foundation (CNS-1116079).

References

[1] H.V. Madhyastha, J.C. McCullough, G.M. Porter, R. Kapoor, S. Savage, A.C. Snoeren, and A. Vahdat, "scc: Cluster Storage Provisioning Informed by Application Characteristics and SLAs," in FAST, 2012.

[2] G.A. Alvarez, E. Borowsky, S. Go, T.H. Romer, R.A. Becker-Szendy, R.A. Golding, A. Merchant, M. Spasojevic, A.C. Veitch, and J. Wilkes, "Minerva: An Automated Resource Provisioning Tool for Large-Scale Storage Systems," ACM Transactions on Computer Systems, 2001.

[3] J. Wilkes, "Traveling to Rome: QoS Specifications for Automated Storage System Management," in IWQoS, 2001.

[4] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using Magpie for Request Extraction and Workload Modeling," in OSDI, 2004.