

Privacy-Sensitive VM Retrospection

Wolfgang Richter[†], Glenn Ammons[‡], Jan Harkes[†], Adam Goode^{*},

Nilton Bila[•], Eyal de Lara[•], Vasanth Bala[‡], Mahadev Satyanarayanan[†]

[†]Carnegie Mellon University, [‡]IBM Research, ^{*}Google, [•]University of Toronto

Abstract

The success of cloud computing leads to large, centralized collections of virtual machine (VM) images. The ability to retrospect (examine the historical state of) these images at a high semantic level can be valuable in many aspects of IT management such as debugging and troubleshooting, software quality control, legal establishment of data or code provenance, and cyber forensics such as malware tracking and licensing violations. In this paper, we explore the privacy implications of VM retrospection. We argue that retrospection will worsen current concerns about privacy in cloud computing. We develop privacy-sensitive requirements for the design of a retrospection mechanism, and then show how they can be met in a functional prototype.

1 Executable Content as Searchable Data

We normally think of VM images as executable content, but the practice of archiving VM snapshots in cloud computing suggests that it may also be valuable to view them as “big data.” Snapshots of a VM image over time represent historical provenance that can be relevant to debugging, troubleshooting, and forensics. We use the term *VM retrospection* for this ability to pose and answer deep questions about VM images. In contrast to VM introspection [5], which examines active VM state during execution, retrospection focuses on passive VM state.

For example, consider a developer who periodically snapshots his VM. When a colleague encounters a misconfiguration bug that has plagued the developer in the past, he is able to search through the VM history to reproduce the bug and to recapitulate its fix. A different example involves a graphic arts company that is accused of copyright infringement. By searching archived VM history of the company’s employees, the plaintiff is able to show how they transformed the original photograph into the disputed product. Other use cases of VM retrospection can be found in a recent position paper [11].

This paper focuses on the privacy implications of VM retrospection. We argue that retrospection will worsen current concerns about privacy in cloud computing. We develop privacy-sensitive requirements for the design of a retrospection mechanism, and then show how they can be met in a functional prototype.

2 Honoring the Principle of Least Privilege

A dominant concern of retrospection is the *need to be extremely sensitive to issues of privacy*. There is already enormous public concern about the potential for compromise of personal privacy in cloud computing. The concentration of VM images in a cloud, rather than the dispersal of that state over myriad personal computers (many inaccessible behind firewalls), simplifies the task of exploring that state for good or evil. Just a small sampling of the huge volume of recent discourse on this topic [2, 3, 7] shows the level of public concern. Microsoft recently requested the United States Congress to pass legislation on privacy in cloud computing [12]. In this highly charged atmosphere, creating a VM retrospection capability is sure to cause alarm. It is therefore essential that the high-level architecture and method of operation of the mechanism be clearly and unambiguously biased in favor of privacy.

The deep concerns about privacy suggest that a good retrospection mechanism is unlikely to be obtained through the most obvious implementation strategy. That strategy would be to leverage the well-understood world of Web search exemplified by Google and Bing, and to use search engine software such as MapReduce to periodically crawl VM images in a cloud and index their content. Such an approach would inflame privacy advocates because of the power that it would give to cloud owners and their search partners. There would be fear of abuse of this power. There would be serious concerns about unauthorized searches by cloud employees on already-built indexes. There would also be great concern over frequent exposure of data for index creation. These issues have not surfaced in the context of Web search because the data in question is, by definition, “published” by their owners. In contrast, VM images in a cloud are not “published” by their owners—they are given for safe-keeping to the cloud operator, much as one places valuables for safe-keeping in a bank’s vault. Since perception matters as much as reality in matters of trust, these privacy-related concerns will be difficult to overcome.

A good retrospection mechanism should be fine-grained and selective in its application. It should involve more heavyweight actions than an index lookup, and there should be an opportunity to inspect and/or audit the computations involved in a search. The mechanism should not require any component (such as a search engine for index creation) to have cloud-wide access to the complete con-

tents of all VM images. Rather, search processing should be performed only as needed, honoring *the principle of least privilege* [10]. Further, the mechanism should support specification of access control policies regarding who can perform retrospection, in what detail, and over what subset of VM images and files within them.

3 Other Requirements for Retrospection

While concern for privacy is the first and most important requirement of a retrospection mechanism, there are a number of other requirements that also need to be taken into account. We examine these next.

Out of an abundance of caution, a VM image owner may encrypt sensitive files but omit their decryption keys from the image. Those files may be decrypted only as needed during the execution of the image, possibly obtaining the keys from runtime user interaction or from a third-party key escrow server that is independent of the cloud. The ability to retrospect such user-encrypted VM content (with user cooperation) emerges as a second requirement. This can be extended to retrospection without user cooperation (e.g., under a search warrant) if key escrow is mandated and enforced. As discussed in Section 5, our architecture gives the user control over the placement of the search engine relative to the cloud that stores VM images. This is a tradeoff between trust and efficiency.

A third requirement for the retrospection mechanism is that it allow *deep content probing* into VM images. Superficial exploration of metadata such as image ownership, modification time, or type of guest operating system won't find "buried skeletons." Even one level deeper is not sufficient: for example, the ability to interpret the file system structure of a virtual disk is necessary, but not sufficient. Just knowing the directory structure and file attributes won't do. One needs to be able to explore the contents of individual files within that file system and to interpret their contents in an application-specific manner. This need for deep content interpretation distinguishes retrospection from previous VM research. That large body of work has focused on low-level aspects of VM execution, management, or monitoring and has treated the guest environment as a black box. In contrast, retrospection focuses on deep examination of guest contents.

The fourth requirement follows from the third: *the ability to use proprietary software in retrospection*. Often, a cloud operator may not possess licenses for the proprietary software needed to interpret some files within VM images stored in the cloud. The transitive closure of all the licenses needed to fully interpret all VM images in a cloud may be substantial. In some cases, the software may be a trade secret of the VM image owner, and simply not available to the cloud operator at any price.

In combination, these requirements strengthen the conclusion we drew earlier on the basis of privacy concerns: a

good retrospection mechanism is unlikely to emerge from a "crawl the cloud and index everything" strategy.

4 Design Principles & Search Model

These considerations lead to the following principles for the design of a good VM retrospection mechanism:

- Principle 1: *A search computation should only be performed on demand for a specific query, and its scope should be restricted to the smallest relevant subset of VM images and files within them.*
This is in contrast to performing search computations *en masse* for index creation in anticipation of future queries. This is essentially the principle of least privilege, as applied to VM retrospection.
- Principle 2: *Control of policy for retrospection should reside with VM owners, not cloud operators.*
The role of cloud operators should be limited to authentication, enforcement of access control policies, and auditing of search computations. This design principle is the cornerstone of personal privacy in retrospection. It gives users confidence that cloud operators cannot subvert their privacy preferences. It also aligns well with future extensions to cloud computing that support portability of VM images: regardless of which cloud a VM image is stored on, its owner can be confident that he controls its retrospection. This design principle also encourages the emergence of standardized access control mechanisms for VM retrospection that are supported by all clouds.
- Principle 3: *There should be as few constraints as possible on the generality of search computations.*
Since file contents on virtual disks can vary over an extremely wide range, from simple text files to highly structured data that can only be parsed by a proprietary application, the range of supported search computations needs to reflect this diversity of data. A design that tightly constrains code for search computations (e.g., requiring it to be written in a specific language) is unlikely to prove satisfactory. It should be possible to create search computations that reflect a high semantic level of interpretation of data.

These design principles have strongly influenced the search model of *Nanuk*, our prototype for VM retrospection. This interactive search model allows creation of customized interfaces for querying different types of content in VM images. A user can view early search results as soon as they are available. Then, based on these early results, he may abort search computations that are still in progress and proceed to refine the query. Melnik et al [8] have shown the importance of iterative refinement for examining large non-VM datasets.

In fulfillment of Principle 1 (least privilege), *Nanuk* performs search computations only after a specific query

| | |
|----------------------|-------------------|
| Number of VMs | 103 |
| Total files | 1.5×10^7 |
| Distinct files | 6.9×10^6 |
| Total bytes in files | 948 GB |
| Dedup disk usage | 489 GB |

Figure 1: File-Granularity Deduplication

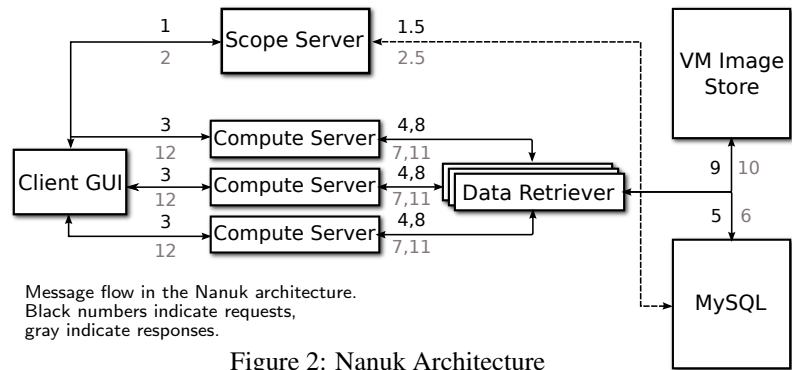


Figure 2: Nanuk Architecture

is posed; however, metadata and other exposed data may be indexed before the query according to user policy. In fulfillment of Principle 2 (privacy control), Nanuk supports search of user-encrypted data. In further fulfillment of Principles 1 and 2, Nanuk allows structured metadata to be used for narrowing the scope of search computations. Such scoping also improves system responsiveness and interactivity. For example, an enterprise may maintain a coarse-grained audit trail of accesses to a collection of highly sensitive VM images in a private cloud. The audit trail may be implemented as a relational database outside the cloud. Retrospection must support searches that span both the structured data of the audit trail and the unstructured file contents within VM images. In fulfillment of Principle 3 (flexibility), Nanuk allows queries to be expressed as executable code on the raw data, rather than being restricted by a declarative query language such as SQL or limited to predetermined indexes. Searches can be performed on the directory structure (i.e., file or directory names and attributes) as well as the file contents of VM disk images.

5 Nanuk Architecture & Implementation

Nanuk builds on *discard-based search*, which was introduced in 2004 by Huston et al [6] for non-text data such as photographs. Consistent with Principle 1, this approach performs search computations only after a specific query is posed. The domain-specific code that performs early discard on servers is called a *searchlet*. It is typically composed of individual components called *filters*. For example, an image search application may provide filters for face detection, color detection and texture detection. Filter code is submitted by the application via a client-side API and is distributed by the runtime system to all of the servers involved in the search task. Each server has a persistent cache of the filter code it has received.

The large size of VMs (typically many GB or tens of GB) is a challenge for Nanuk. Discard-based search involves runtime disk I/O to read the contents of each VM image, and processing overhead to apply a searchlet to this large amount of data. For a search of large scope, the

total runtime I/O and processing can be intolerably large. Nanuk exploits similarity of data content across VM images as a performance optimization. A file that recurs in many VM images only needs to be examined once in any search. Note that this benefit is in addition to the well-known storage savings achievable through deduplication.

Nanuk exploits data similarity at the granularity of individual files within the virtual disks of VM images, rather than at block level. Due to the different sizes of VM images, different software installation histories, and different execution histories, the mapping of identical files to blocks may be different in different VM images. Similarity at the file level is then lost at the block level. Nanuk deconstructs and stores VM images along the lines described by Reimer et al [9]. The deconstruction is performed when a VM image is initially deposited into a cloud. Modifications to VM images are efficiently tracked using copy-on-write techniques at whole-file granularity. On-demand reconstruction of a full VM image from its constituent files is efficient.

Our preliminary results indicate that whole-file deduplication is effective in Nanuk. Figure 1 shows the data characteristics of a collection of 103 VM images in our prototype cloud: 65 Windows XP VMs (each loaded with different applications) from the VCL cloud at North Carolina State University [13], and 38 Linux VMs (different distributions loaded with unique research datasets) from Carnegie Mellon University. Out of a total of 15 million files occupying 948 GB, there were only 6.9 million distinct files requiring 489 GB of disk space.

Figure 2 shows the Nanuk architecture. The file-level contents of deconstructed VM images, addressed by the cryptographic hashes of files, are stored in the component labelled “VM Image Store.” During deconstruction, which happens only once for each VM image, a considerable amount of meta-data is generated. This includes standard file system meta-data plus contextual information about the VM images in which the files occur. Since this meta-data can be valuable for narrowing the scope of a discard-based search, it is persistently stored




in MySQL. Code components called *data retrievers* provide the search engine with access methods to the data.

As mentioned earlier, VM retrospection may involve composite queries that combine indexed search on structured data (such as an audit trail) with discard-based search of unindexed file contents. The box labelled “Scope Server” in Figure 2 is the bridge between the indexed and unindexed worlds. A user must authenticate to the scope server at the start of a search session. The structured part of a composite query (called a “scope request”) is first processed by the scope server on MySQL. The result is encoded into a cryptographically signed token called a *scope cookie*, which is essentially a capability for the subset of files in the VM image store that are within scope for the upcoming discard-based search.

Figure 2 is best understood by following the message flows that occur during a search. We first describe retrospection of unencrypted data. A user defines the scope of a Nanuk search using a Web browser (labeled “Client GUI” in Figure 2) (1). The scope server performs access control by contacting MySQL (1.5,2.5), then constructs a scope cookie and returns it to the client (2). The client then presents the cookie to the compute servers (3). The compute servers validate the cookie and pass it along to the data retriever(s) (4). A data retriever transforms the scope request into a SQL query, then presents it to MySQL (5) to identify the list of files that are within scope, and returns the result (6) to the compute servers (7). The distribution of search computations across compute servers happens implicitly at this point. The compute servers perform early-discard on files within scope, using a data retriever to obtain the files from the VM image store (8,9,10,11). If a file is not discarded, a low-fidelity version (i.e., “thumbnail”) is returned to the client (12). Its full-fidelity version can be obtained on demand.

The placement of compute servers and data retrievers in Figure 2 represents a tradeoff between efficiency and trust. A paranoid user can choose to place these entities on a different cloud from the one that stores his VM images. The latter cloud then never sees his files in decrypted form. More typically, a user will have sufficient trust in the cloud that stores his VM images to colocate the compute servers and data retrievers. This is much more efficient because it avoids data transfer between clouds. This placement decision does not have to be made a priori, but can be deferred to the point at which the search is performed.

We are exploring three distinct architectures for retrospecting user-encrypted data: (1) one in which users trust the cloud, (2) one in which users trust the cloud and a key escrow service, and (3) one in which users trust no third party with encryption keys. As a simple initial implementation, we allow the user to provide decryption keys in the scoping step (i.e., directly to the scope server as part of step (1) in Figure 2). This takes the form of a list of 3-

| | Search Task | Sample Image | Search Statistics |
|---|---|---|---------------------|
| 1 | Find 10 images of beaches. |  | 12140 / 11395 / 681 |
| 2 | Given an image of a crocodile with a tree, find at least one similar image. |  | 20427 / 20418 / 7 |
| 3 | Given an image of a dog, find 3 similar images. |  | 20933 / 20365 / 560 |

The last column shows the number of images processed by the servers, the number discarded, and the number shown to the user.

Figure 3: Trace tasks and user results

tuples: $\langle \textit{pathname}, \textit{encryption method}, \textit{key} \rangle$. This list is used by data retrievers to decrypt files before they are presented to compute servers (i.e., just before step (11)). The compute servers never see encrypted data. The data retrievers flush keys after decrypting the associated files.

We are working to extend this mechanism to be less verbose and cumbersome for searches of large scope. This requires striking a balance between usability, privacy and performance. At one extreme is a single encryption key for an entire VM image. The other extreme (our initial choice) is a key per file within a VM image. A hierarchical file system within a VM image offers natural directory-level or subtree-level aggregation possibilities for intermediate points of this spectrum. This would require augmenting the 3-tuples mentioned above with an element denoting the granularity of the decryption key: $\langle \textit{granularity}, \textit{pathname}, \textit{encryption method}, \textit{key} \rangle$, with possible granularity values of “VM image,” “subtree,” or “file.” We also plan to investigate use of an external key escrow service. Use of such a service could improve usability and simplify conformance with enterprise-wide data security and audit policies. Instead of providing a list of 3-tuples or 4-tuples in step (1), the user would now need to provide sufficient information for later use of the key escrow service by data retrievers during step (11). Depending on the latency and throughput of the escrow service, the data retrievers may need to prefetch keys during discard-based search in order to avoid stalls. In addition, we are exploring the option of giving users direct control of the search infrastructure in the cloud for their searches such that they never reveal keys to any other party. This final path maintains privacy while providing a service that scales with the scope of searches.

6 Status and Preliminary Evaluation

At present, Nanuk is a proof-of-concept prototype. Nanuk supports three applications: (a) HyperFind, which enables users to interactively search photographs in common formats such as JPEG, TIFF and BMP; (b) ShingleFind, which searches for files whose content approximately matches an example text using the tech-

| Servers | Trace 1 | | Trace 2 | | Trace 3 | |
|---------|---------|-------|---------|-------|---------|-------|
| | Native | Nanuk | Native | Nanuk | Native | Nanuk |
| 1 | 988 | 948 | 1284 | 1325 | 1265 | 1274 |
| 2 | 549 | 549 | 678 | 680 | 743 | 743 |
| 4 | 334 | 334 | 375 | 376 | 480 | 483 |
| 6 | 260 | 261 | 275 | 274 | 396 | 393 |
| 8 | 225 | 231 | 226 | 225 | 355 | 356 |

Each data point is the mean of three runs. All standard deviations are less than 6% of the mean.

Figure 4: Task completion time (Seconds)

nique of *w-shingling* [1]; and (c) ClamFind, which searches for viruses and potential vulnerabilities in files using the ClamAV engine [4].

Many improvements to the performance and scalability of Nanuk are yet to be implemented. For space, and because optimizations are currently being implemented, we omit early results from, for example: an examination of Nanuk’s ability to exploit server parallelism, an investigation of its limits on scalability with respect to number of VM images stored, its performance sensitivity to storage technologies and storage layout, and its sensitivity to network bandwidth. However, even in Nanuk’s early state, we are able to answer one important question: *Can users retrospect VM images as effectively as they can search server files?* In other words, what is the impact of the architecture shown in Figure 2 on the interactive response seen by a user during a search?

To answer this question, the HyperFind application was used to capture and replay the three traces summarized in Figure 3. For each task, the user was given a time limit of 20 minutes. The search could be targeted at photographs spread over the local file systems of a collection of 65 Windows XP VM images in Nanuk (labeled “Nanuk” in our results), or they could be targeted at the same set of photos spread over the local file systems of the compute servers shown in Figure 2 (labeled “Native” in our results). In both cases the hardware was identical: The compute servers were Dell Vostro 220’s, each with an Intel Core 2 Duo 3 GHz processor, 3 GB of memory, and a 7200 RPM SATA drive. A single physical machine hosted the VM image store, the MySQL database, and the data retriever. This machine had two Quad Core Intel Xeon 2.66 GHz processors, 32 GB of memory, and a fibre channel RAID5 with 12 10K RPM drives. The client was a 2.66 GHz Intel Core 2 Duo Apple iMac with 4 GB of memory. All network connections were Gigabit Ethernet.

For each trace, Figure 4 presents the time for task completion. For all traces, as the number of compute servers increases, the task completion time decreases. This is because more computational resources are being devoted to the search. For each trace and number of compute servers, there is at most a small increase in the task completion times on Nanuk, relative to the native case. The worst case difference is about 3%. In most cases, the times are identical within bounds of experimental error. Fig-

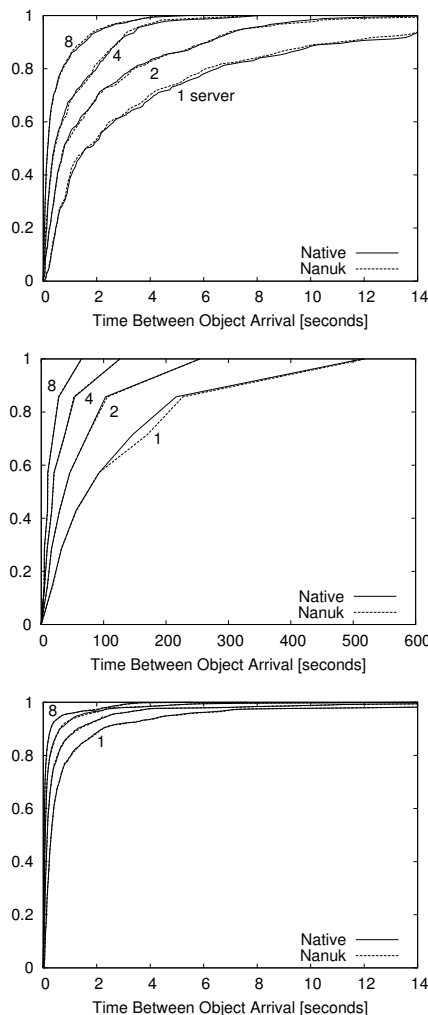


Figure 5: Traces 1, 2 and 3 object interarrival time CDFs

ures 5a, 5b, and 5c show the CDFs of observed result interarrival times for Traces 1, 2, and 3. The curves for the native and Nanuk cases are close, confirming that interactive user experience is virtually indistinguishable between the two cases. These results confirm a user can hardly tell the difference between searching local files on servers or retrospect data within VM images.

7 Conclusion

In this paper, we motivated four key requirements that must be satisfied by a mechanism for VM retrospection. From these requirements, we derived a set of design principles for a retrospection system. Foremost among these principles is respect for user privacy, achieved by honoring the principle of least privilege. Guided by these principles, we have designed and implemented a prototype system for retrospection providing us with early validation and feedback.

Acknowledgements

This research was supported by the National Science Foundation (NSF) under grant number CNS-0833882, an NSF Graduate Research Fellowship, and an IBM Open Collaborative Research grant. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily represent the views of the NSF, IBM, or Carnegie Mellon University.

References

- [1] BRODER, A., GLASSMAN, S., MANASSE, M., AND ZWEIG, G. Syntactic clustering of the web. In *Proceedings of the 6th International WWW Conference* (1997).
- [2] BRUENING, P. J., AND TREACY, B. C. Privacy, Security Issues Raised by Cloud Computing. *Privacy and Security Law Report 8 PVL R 10* (March 2009).
- [3] CAVOUKIAN, A. Privacy in the Clouds. *Identity in the Information Society 1*, 1 (December 2008), 89–108.
- [4] CLAMAV. Clam AntiVirus. <http://www.clamav.net/>.
- [5] GARFINKEL, T., AND ROSENBLUM, M. A Virtual Machine Introspection-Based Architecture for Intrusion Detection. In *Proc. 10th Symp. Network and Distributed System Security* (2003).
- [6] HUSTON, L., SUKTHANKAR, R., WICKREMESINGHE, R., SATYANARAYANAN, M., GANGER, G.R., RIEDEL, E., AILAMAKI, A. Diamond: A Storage Architecture for Early Discard in Interactive Search. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies* (San Francisco, CA, April 2004).
- [7] LARKIN, E. Will Cloud Computing Kill Privacy? *PC World* (January 2010).
- [8] MELNIK, S., GUBAREV, A., LONG, J. J., ROMER, G., SHIVAKOMAR, S., TOLTON, M., AND VASSILAKIS, T. Dremel: Interactive Analysis of Web-Scale Datasets. In *The 36th International Conference on Very Large Data Bases* (September 2010), vol. 3.
- [9] REIMER, D., THOMAS, A., AMMONS, G., MUMMERT, T., ALPERN, B., AND BALA, V. Opening Black Boxes: Using Semantic Information to Combat Virtual Machine Image Sprawl. In *VEE'08: Proceedings of the 2008 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (Seattle, WA, March 2008).
- [10] SALTZER, J. H. Protection and the Control of Information Sharing in Multics. *Communications of the ACM 17*, 7 (1974).
- [11] SATYANARAYANAN, M., RICHTER, W., AMMONS, G., HARKES, J., AND GOODE, A. The Case for Content Search of VM Clouds. In *Proceedings of the First IEEE International Workshop on Emerging Applications for Cloud Computing (CloudApp 2010)* (Seoul, South Korea, July 2010).
- [12] SMITH, B. (SENIOR VICE PRESIDENT AND GENERAL COUNSEL, MICROSOFT). Cloud Computing for Business and Society. Keynote Speech, Brookings Institution, January 20, 2010, available at http://www.microsoft.com/presspass/presskits/cloudpolicy/docs/20100120_transcript.pdf.
- [13] VOUK, M., RINDOS, A., AVERITT, S., BASS, J., BUGAEV, M., PEELER, A., SCHAFFER, H., SILLS, E., STEIN, S., THOMPSON, J., AND VALENZISI, M. Using VCL technology to implement distributed reconfigurable data centers and computational services for educational institutions. *IBM Journal of Research & Development 53*, 4 (September 2009).