CASL: Computer Architecture and Systems Laboratory

Department of Computer Science and Automation,Indian Institute of Science, Bangalore

Visit us at: http://drona.csa.iisc.ernet.in/~casl/

Prudent Memory Reclamation in Procrastination based Synchronization

Procrastination is the fundamental technique used in highly scalable synchronization mechanisms such as Read-Copy-Update (RCU). In this technique writers defer the freeing of an object until there are no readers referring to the object.

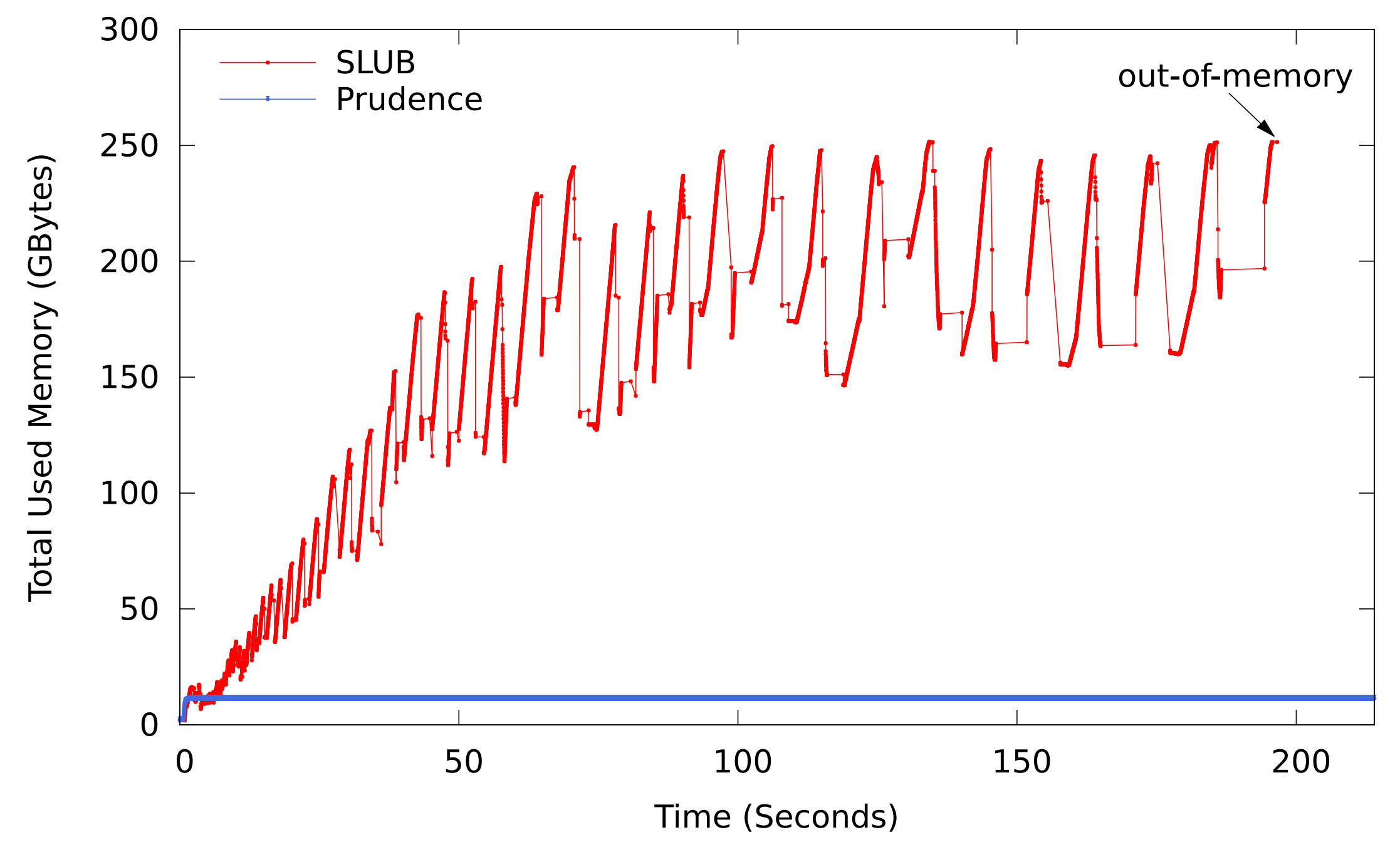

Figure 1: Impact of RCU on the SLUB allocator in Linux Kernel

Problem Statement

We observe that the deferred freeing of the objects by the procrastination-based synchronization mechanism induces poor memory allocator performance.

Our Work: We develop Prudence, a dynamic memory allocator, which is tightly integrated with the synchronization mechanism. The Prudence memory allocator looks at procrastination from the perspective of providing hints into the future and exploits such hints to improve the performance of the memory allocator.

References:

1. Prasad, A. and Gopinath, K. (2016). Prudent memory reclamation in procrastination-based synchronization. In Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '16, pages 99–112, New York, NY, USA. ACM.

Illuminator: Fighting Memory Fragmentation

Complex and large working sets of modern applications are known to gain substantial benefits with huge pages. Huge pages can minimize the Virtual to Linear address translation cost by mapping large portions of process address space into a single TLB entry. However, facilitating allocation of large contiguous blocks is a challenge for operating systems because of fragmentation.

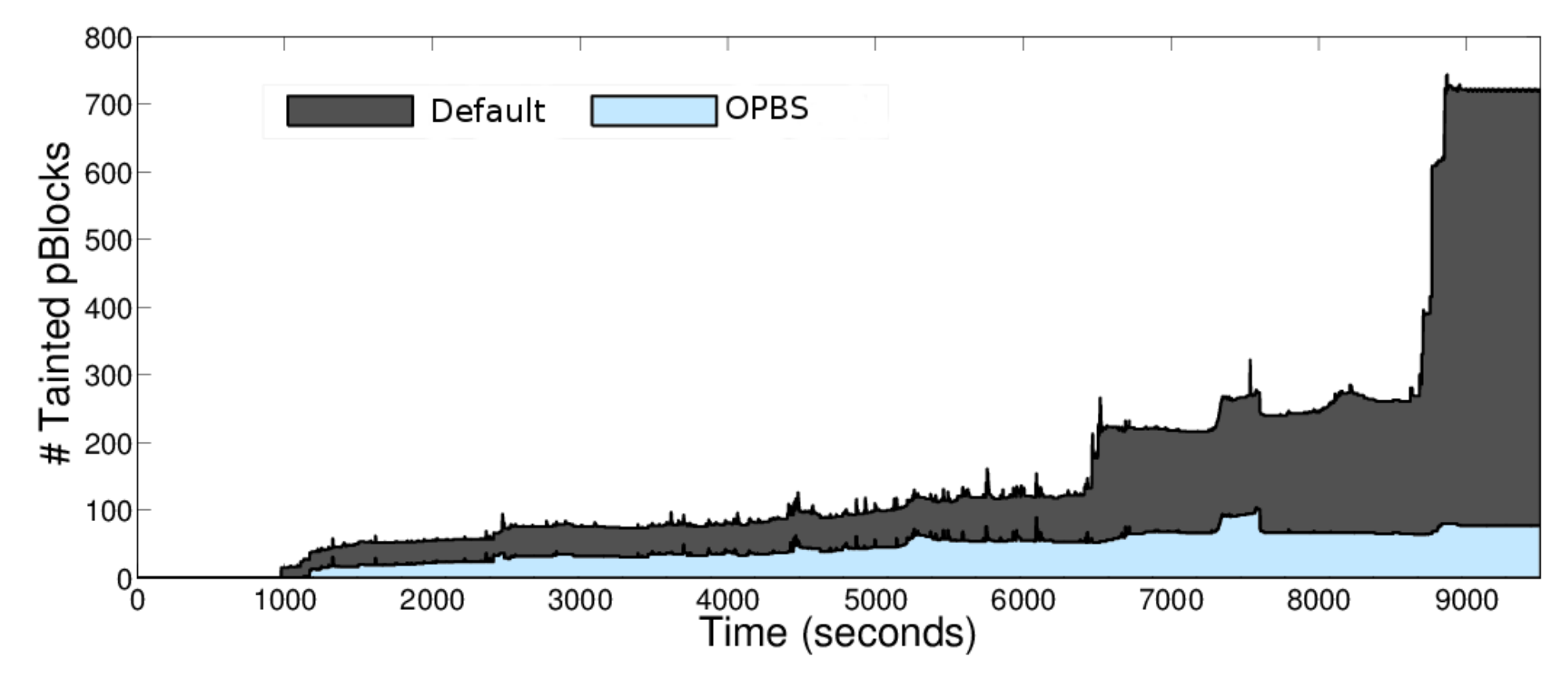

Figure 2: Rate of Pollution of Huge Pages

Problem Statement

Typically all operating systems have unmovable kernel pages that hinders compaction which is used to tackle fragmentation. Scattering of kernel pages in physical address space is major concern in fighting fragmentation. This in turn restricts the availability of Superpages to user application.

Our Work: Proposed new allocator which reduces the scattering of kernel pages by introducing a new class of superpages called composite superpages. Allocates 2x to 43x more superpages. Reduces de-fragmentation effort by 89%. Improves the overall performance of user application up to 36%.

References:

1. Panwar, A., Patel, N., and Gopinath, K. (2016). A case for protecting huge pages from the kernel. In Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems, APSys '16, pages 15:1–15:8, New York, NY, USA. ACM.

Hadoop Performance Optimization

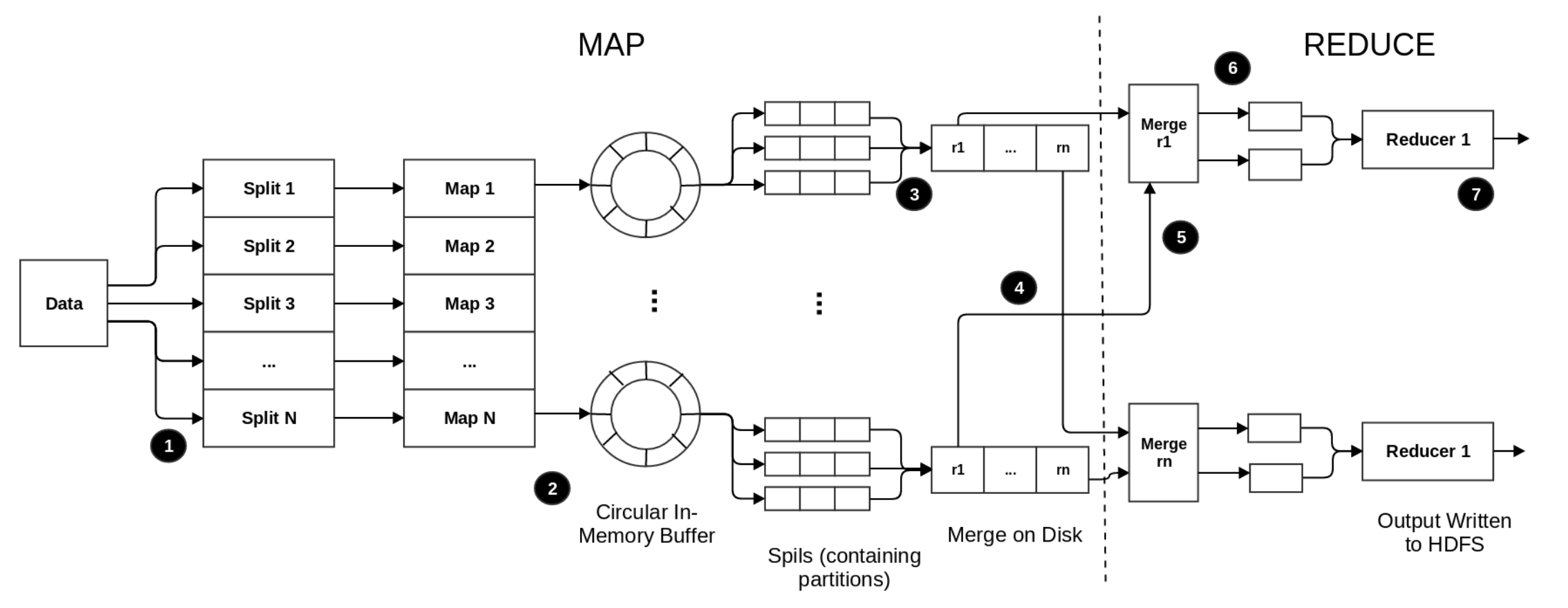

Figure 3: MapReduce: Distributed and Parallel Processing

Hadoop MapReduce is a framework used for distributed storage and processing of large datasets.

Problem Statement

Hadoop has various configuration parameters which affect the performance (execution time) of a given big data processing job. Default values of these parameters do not result in good performance and hence it is important to tune them. However, there is inherent difficulty in tuning the parameters - the parameter search space is large and added to it there are cross-parameter interactions. Thus, we need a dimensionality-free method which can automatically tune the configuration parameters by taking into account the cross-parameter dependencies.

Our Work: We propose a novel Hadoop parameter tuning methodology, based on a noisy gradient algorithm known as the simultaneous perturbation stochastic approximation (SPSA). The SPSA algorithm tunes the selected parameters by directly observing the performance of the MapReduce system. Our method, when tested on a 25 node Hadoop cluster shows 45-66% decrease in execution time of Hadoop jobs on an average, when compared to the default configuration and prior methods.

References:

1. Sandeep K, Sindhu P R, Priyank P, K. Gopinath, and S Bhatnagar. Performance Tuning of Hadoop MapReduce: A Noisy Gradient Approach. arXiv preprint arXiv:1611.10052 (2016).

An Information flow based microkernel

The standard security policies, such as firewalls and access control mechanisms, are not enough to prevent information leakage. Information flow model limits information leaks via overt and covert channels. Labelling Mechanism can be used to model Information flow control. Information can flow from an object labelled L1 to another object labelled L2 only if L2 is at least as tainted as L1 in every category.

Problem Statement

To enhance the security of a microkernel using information flow. Information flow is enforced using **Histar labels** (Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazi'eres. Making information flow explicit in histar. In Proceedings of the 7th symposium on Operating systems design and implementation, USENIX Association, 2006.).

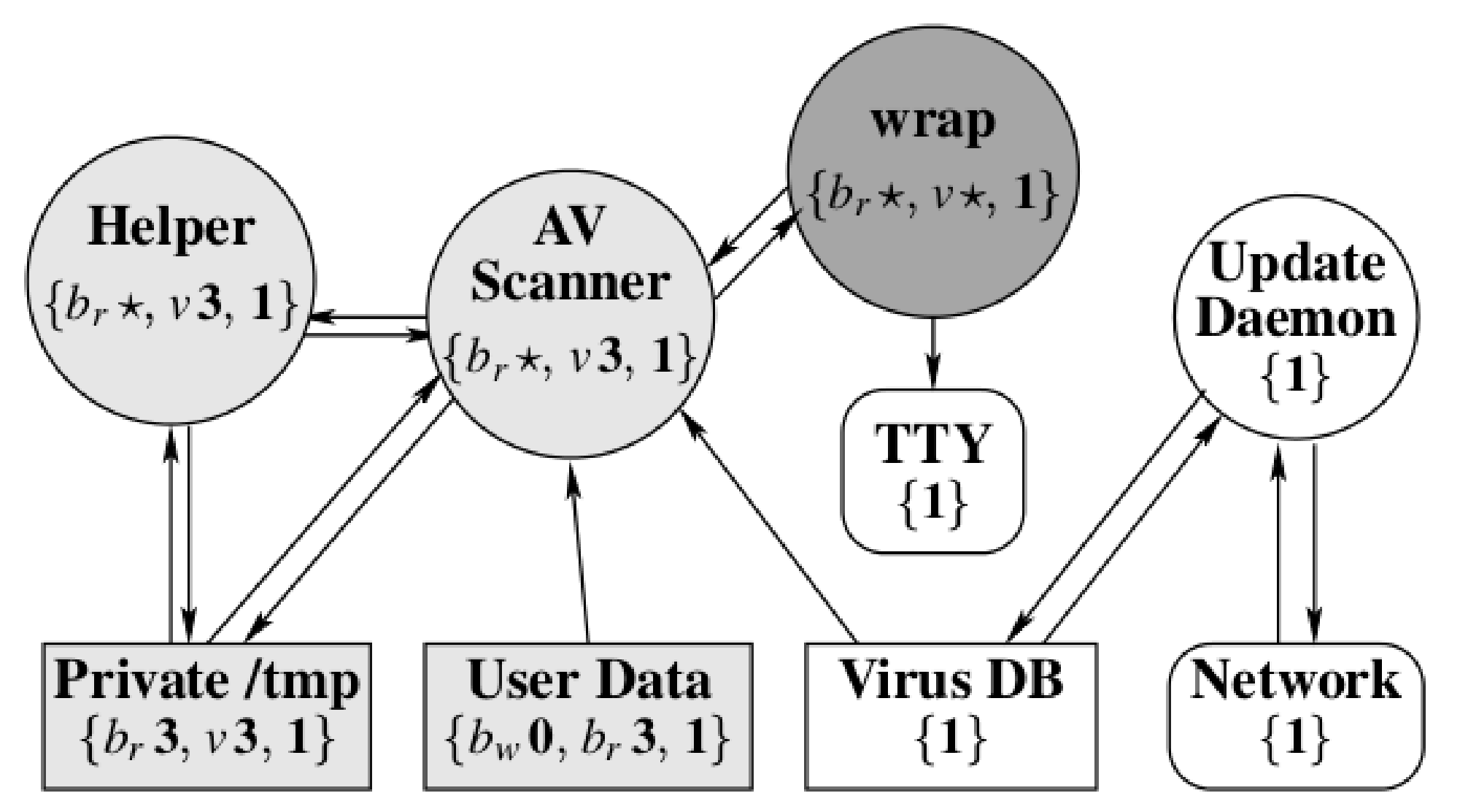
Helper $\{b_r\star, v3, 1\}$
AV Scanner $\{b_r\star, v3, 1\}$
wrap $\{b_r\star, v\star, 1\}$
Update Daemon $\{1\}$
TTY $\{1\}$
Private /tmp $\{b_r3, v3, 1\}$
User Data $\{b_w0, b_r3, 1\}$
Virus DB $\{1\}$
Network $\{1\}$

Figure 4: Labels associated with antivirus components to protect information leakage.[Making information flow explicit in histar]

Our Work: Implemented a Label Server to track the information flow in MINIX 3.3. Necessary system calls to make use of the Label constructs were also provided. Label Server starts with 16 worker threads, which grows and shrinks based on the workload. Label server is persisted to a main backing file after a fixed number of modifications are made to the Labels. Labels are used to protect the Label Server resources.