國 立 清 華 大 學

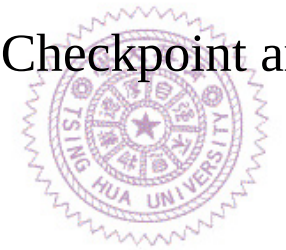
資訊工程研究所

碩士論文

使用中斷點和回復機制

實現容器叢集中的遷移和高可用性

Container Migration and High Availability in Docker Swarm using Checkpoint and Restoration



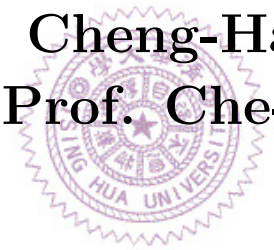
研究生：103062622 黃晟豪（Cheng-Hao Huang）

指導教授：李哲榮 教授（Prof. Che-Rung Lee）

中華民國一零五年七月

Container Migration and High Availability in Docker Swarm using Checkpoint and Restoration

Student: Cheng-Hao Huang
Advisor: Prof. Che-Rung Lee



Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan, 30013, R.O.C.

July 2016

中文摘要

容器技術自 2013 年 Docker 發表後在全世界迅速竄紅，Container 解決了維護人員在伺服器進行大量部屬時的痛點，使得環境部屬只需要建立完容器映像檔後就可以進行大量部屬，並對每一個容器環境進行隔離。

在這篇論文中，我們提出了在 Docker swarm 叢集中，將容器在多個節點中相互搬移。另外，可以針對特定的容器定期設定 checkpoint 儲存至雲端儲存空間，若叢集中的節點遇到不正常的離線時，可以及時回復最近的容器狀態到健康的節點上。

Abstract

Contents

Chinese Abstract i

Abstract i

Contents iv

List of Figures vi

List of Tables vii

List of Algorithms viii

1 Introduction 1

2 Background 3

2.1 Docker . . . 3

2.1.1 Docker Client . . . 3

2.1.2 Docker Daemon . . . 4

2.1.3 runC . . . 4

2.2 Docker Swarm . . . 4

2.2.1 Discovery services . . . 5

2.2.2 Scheduler . . . 6

2.2.3 High availability of Swarm Manager . . . 7

2.2.4 High availability of Docker Swarm containers . . . 7

2.3 CRIU . . . 7

3 Design and Implementation 8

3.1 Docker . . . 8

3.1.1 Docker Client . . . 8

3.1.2 Docker Daemon 9
3.2 Docker Swarm configuration 9
3.3 Docker containers migration in Docker Swarm 9
3.4 Docker Swarm checkpoint and restoration rescheduling policy 11
3.4.1 Docker Swarm container checkpoint tickers 11
3.4.2 Docker Swarm restore rescheduling policy 12
3.4.3 High availability of Swarm Manager in Docker Swarm checkpoint and restoration rescheduling policy 13

4 Experiments **15**

5 Related Work **16**

6 Conclusion **17**

List of Figures

2.1 Single node Docker 3
2.2 Docker Swarm architecture 5

3.1 Docker Swarm with remote storage server 10
3.2 Containers checkpoint versions in remote storage server 13

List of Tables

List of Algorithms

1 Checkpoint ticker algorithm 12
2 Restore rescheduling policy algorithm 14

Chapter 1

Introduction

Traditionally, people rely single supercomputer to calculate data or deploy their companies's application. Nowadays, cloud computing has been use wildly. People would like to use a lot of normal computers that gather them into a cluster to replace a supercomputer. More and more companies build their data centres or use cloud platforms to construct their business application. However, the more computers we have, the more power consumption problem we have to solve. At the same time, to make sure every computers' process in the cluster are alive, high availability becomes a more important role in cloud computing.

Virtualization is a popular technology that is used wildly on cloud computing, including virtual operating system, computer hardware platforms, storage device and computer network resources. Virtual machine is a technology to emulate a particular computer like a real computer. It can partition a physical machine resource, such as CPU, memory, storage, and network. A hypervisor uses execution to manage and share host physical machine hardware, it allows many different virtual machines isolated from each other.

Container is operating system level virtualization which runs as an isolated process in userspace on the host operating system and shares the same operation system kernel with other containers. It provides kernel namespace such as PID, IPC, network, mount, and user namespace to isolate each container environment to host operation system. In order to control hardware resource like CPU, memory, network, and disk I/O, container uses cgroups to limit each container resources. Container doesn't have hypervisor to isolate with host operating system, therefore, it can offer better

performance than virtual machine.

Checkpoint and restoration can freeze a running process state and save process information to checkpoint images. User can use these checkpoint image files to restore the process which user want to restore the process state. These features can be used to dump checkpoint and restore containers because each containers are processes in the host operating system.

In this paper, we not only use checkpoint and restoration to migrate containers between two physical computers but also migrate containers in the Docker swarm cluster. Moreover, we improve high availability and rescheduling feature in Docker Swarm using checkpoint and restoration than keep versions of container checkpoint images in remote storage server that whenever computer are fail, Docker Swarm manager will restore the containers from the checkpoint images.

Chapter 2

Background

2.1 Docker

Docker [2] is a open-source project container engine. It provides an additional layer of abstraction and automation of operating-system-level virtualization in Linux. Besides, Docker has extra image management and layered file system to reduce disk space. Docker has two parts, including Docker Client and Docker Daemon.

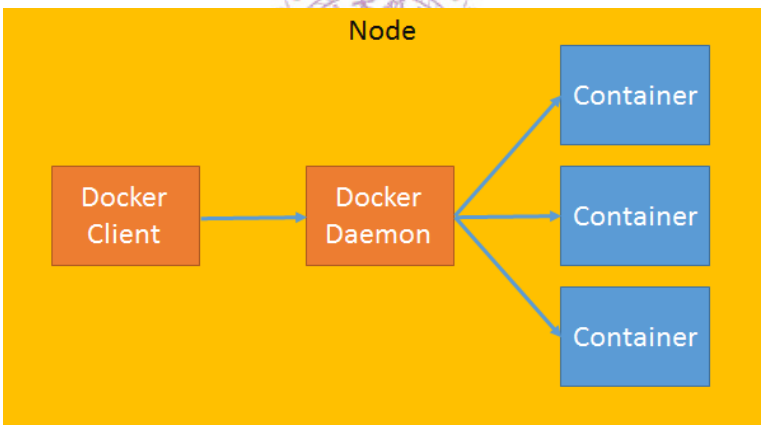


Figure 2.1: Single node Docker

2.1.1 Docker Client

Docker is typical client/server architecture application. Docker Client is a CLI(Client Line Interface) in Docker. Docker uses Docker Client to send and receive requests to Docker Daemon. Also, Docker supports remote RESTful API to

send and receive HTTP requests to Docker Daemon. In additional, it has been implemented by more than 10 programming languages.

2.1.2 Docker Daemon

Docker Daemon is a daemon that runs as system service. It has two the most importance features:

- Receive and handle requests from Docker Clients.
- Manage containers.

When Docker Daemon is running, it will run a server that receives requests from Docker Clients or remote RESTful API. After receiving requests, server will pass the requests by router to find the handler to handle the requests. By default, Docker Daemon listens UNIX socket requests, it serves root permission, or docker group membership. Whenever user wants Docker Daemon to listen remote RESTful API or Docker Swarm requests, it has to enable the TCP socket.

2.1.3 runC

runC is a CLI tool for running containers according to the OCI(Open Container Initiative) specification. It doesn't need any dependency from the operating system, it can control Linux Kernel include namespace, cgroups, apparmor, network, capabilities, firewall, etc. runC provides a standard interface to support the containers management that Docker can use it to control the containers.

2.2 Docker Swarm

Docker Swarm [3] is a native clustering for Docker. It gathers several Docker engines together into one virtual Docker engine. Docker Swarm serves standard Docker API, so it can be connected by Dokku, Docker Machine, Docker Compose, Jenkins, DockerUI, Drone, etc. It also support Docker Client of course.

In Docker Swarm, it has two components which are Swarm Manager and Swarm Node. Swarm Manager is the manager which handles Docker Client and RESTful API requests and manages multiple Docker Nodes resources. Docker Node is an

agent which sends heartbeat to Discovery Service to ensure Docker Daemon is alive in the cluster.

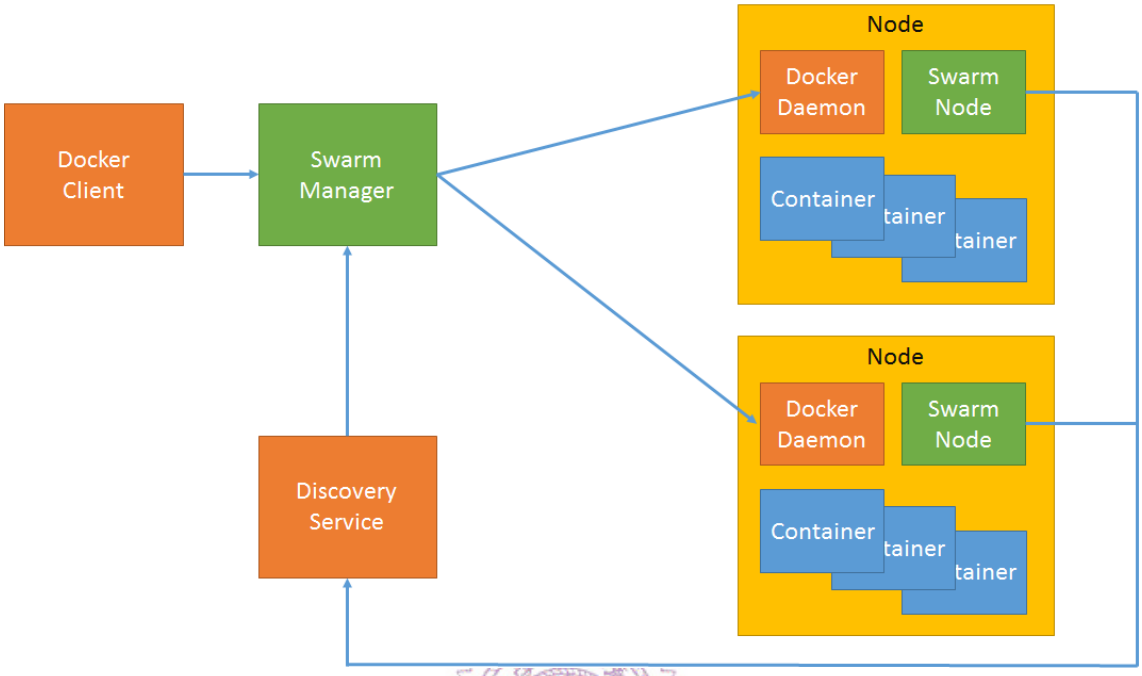


Figure 2.2: Docker Swarm architecture

2.2.1 Discovery services

Docker Swarm provides multiple Discovery Services backends. They are used to discover the nodes in the cluster. There are:

- Using a distributed key/value store, like Consul, Etcd and Zookeeper.
- A static file or list of nodes.
- Docker Hub as a hosted discovery service.

Otherwise, it also supports any modules which satisfy Discovery API interface.

2.2.2 Scheduler

Docker Swarm scheduler decides which nodes to use when creating and running a container. It has two steps. First, It follows user's filters to decide which nodes are conform. Second, It passes through strategies to select the best node in the cluster.

Filter

Filters are divided into two categories, node filters and container configuration filters. Node filters operate on characteristics of the Docker host or on the configuration of the Docker Daemon. Container configuration filters operate on characteristics of containers, or on the availability of images on a host. The node filters are:

- Constraint
- Container slots
- Health filter

The container configuration filters are:

- Affinity
- Dependency
- Port filter

Strategies

The Docker Swarm scheduler features multiple strategies for ranking nodes. Docker Swarm currently supports these values:

- Spread
- Binpack
- Random

Spread and Binpack strategies compute rank according to a nodes available CPU, its RAM, and the number of containers it has. It selects a node at random. Under

the Spread strategy, Swarm chooses the node with the least number of containers. The Binpack strategy chooses the node which has executed most containers. The Random strategy uses no computation, chooses nodes at random regardless of their available CPU or RAM.

2.2.3 High availability of Swarm Manager

In Docker Swarm, Swarm Manager responses the cluster and manages the resources of multiple Docker Nodes at scale. If Swarm Manager dies, we have to create a new one and deal with the interruption of service.

The High availability feature allows Docker Swarm has multiple Swarm Manager instances. We can create a primary manager and multiple replica instances. Whenever we send requests to replica instances, it will be automatically proxied to the primary manager. In addition, if the primary manager fails, the others replica instances will lead a new primary manager.

2.2.4 High availability of Docker Swarm containers

In Docker Swarm, it has rescheduling policy. As we set the reschedule policy when we start a container, whenever Swarm nodes fail, Swarm Manager will restart all of the containers which on the fail nodes to another alive Swarm Nodes.

2.3 CRIU

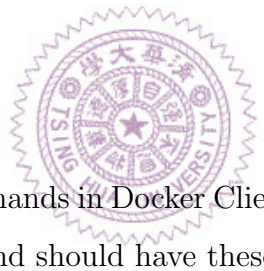
CRIU [1] (Checkpoint/Restore in Userspace) stands for Checkpoint and Restore in User Space, creates a complete snapshot of the state of a process, including things like memory contents, file descriptors, and even open TCP connections. It can be used for suspending and resuming processes, or live migrating them from one machine to another.

Chapter 3

Design and Implementation

3.1 Docker

In native Docker, It has two part, Docker Client and Docker Daemon. Docker Daemon has many components include server, engine, registry, graph, driver and runC. To support dump checkpoint and restore request, some of these steps should be implemented.



3.1.1 Docker Client

We implement 3 Docker commands in Docker Client, including checkpoint, restore, migrate. In checkpoint command should have these configurations:

- image directory - Dump checkpoint image directory.
- work directory - Dump checkpoint image log directory.
- leave running - After dumping checkpoint image, keeping running container or not.
- pre-dump - Pre-dump checkpoint memory image to minimize frozen time.
- pre image directory - Define which version image to compare.
- track memory - Track memory to pre image directory image to minimize disk space.

In restore command should have these configurations:

- image directory - Checkpoint image directory to restore from.
- work directory - Directory for restore log.
- force - Fore restoring container from image directory whether container is running or not.

In migrate command, it focus on Docker Swarm Scheduler filter configurations. In run command, we can do this with the environment variable or the label. Therefore, we implement environment variable and the label configurations in migrate command.

3.1.2 Docker Daemon

In native Docker Daemon, it doesn't support checkpoint and restore command. Fortunately, it is already implemented in runC, so we have to add a proxy between Docker Daemon and runC, which can handle Docker Client's checkpoint and restore requests.

3.2 Docker Swarm configuration

As Figure 3.1, we prepare a remote storage server for saving Docker containers dump checkpoint images. It should have fault tolerant to avoid service shout down. For these reasons, We choose glusterFS to be our experiments remote storage server.

After setting up glusterFS, we mount it to every Docker nodes in the same folder. To avoid mount it as absolute path in every Docker nodes, we mount it to Docker root which we can change configuration in Docker Daemon.

3.3 Docker containers migration in Docker Swarm

Docker Swarm creates containers through Swarm scheduler to dispatch Docker nodes. If we want to specific assign which node we want to create containers, we have to set filters like constraint, affinity or dependency. To migrate containers in Docker Swarm, we must avoid the containers which we want to migrate that migrate to an other nodes, instead of the same node.

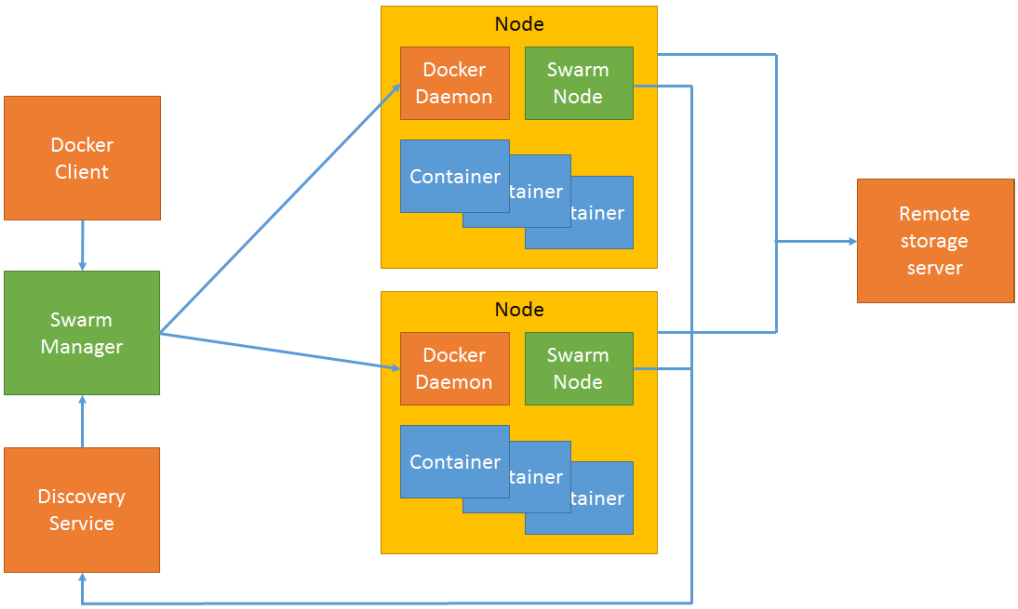


Figure 3.1: Docker Swarm with remote storage server

Step 1. Check Docker Swarm cluster has at least two Swarm nodes.

Step 2. Parse Docker Client requests to analyse label and environment variables, and transform label and environment variables to Docker Swarm filters.

Step 3. Add constraint filter to make sure the container which we want to migrate does not migrate to the same node.

Step 4. Pre-dump the container checkpoint image which we want to migrate to decrease container frozen time.

Step 5. Dump the container checkpoint image by tracking memory from pre-dump checkpoint image.

Step 6. Create empty container on the Docker Swarm scheduler chooses node.

Step 7. Restore the container to the Docker Swarm scheduler chooses node.

Step 8. Delete the checkpoint images.

Step 9. If the container was migrated which had set checkpoint restore rescheduling policy, it will restart checkpoint restore rescheduling policy 3.4.

3.4 Docker Swarm checkpoint and restoration rescheduling policy

In Docker Swarm, it has rescheduling policy. As we set the reschedule policy when we start a container, whenever Swarm nodes fail, Swarm Manager will restart the containers which on the fail nodes to another alive Swarm Nodes.

We improve this policy that we dump the checkpoint image for every containers which we want to keep the container checkpoint for every containers checkpoint ticker pounding. Whenever Swarm Nodes fail, Swarm Manager will restore the containers which Swarm Manager has dumped the checkpoint. Otherwise, the checkpoint tickers policy provides version of checkpoint image by tracking memory. It only dump different memory page checkpoint to new version checkpoint image.

In addition, it also support high availability that whenever Docker Swarm primary manager fails, the others Swarm Manager replica instances will lead a new primary manager. After replica leading a new primary manager, it will restart container checkpoint tickers.

3.4.1 Docker Swarm container checkpoint tickers

Step 1. Set checkpoint-time label and version-group label when we create the container.

Step 2. Swarm Manager analyzes checkpoint label when the container has created. If version-group label doesn't set, version-group will be set to 5.

Step 3. After the container starting, starting the container checkpoint ticker. Container checkpoint tickers are pounding when users want to checkpoint container repeatedly at regular intervals. Container checkpoint tickers will do these steps:

Step a. Swarm Manager sends pre-dump checkpoint request to Swarm Node's Docker Daemon when every Pre-dump version start.

Step b. After pre-dumping the container, Swarm Manager sends dump new version checkpoint request to Swarm Node's Docker Daemon. Every new checkpoint images track memory to last checkpoint version

image, just save memory difference in new checkpoint image. We save version-group(default 5) versions in the same directory(Figure 3.2).

Step c. Send delete checkpoint request to Swarm Node's Docker Daemon to delete oldest Pre-dump version directory whenever container has more than two Pre-dump versions directory.

Algorithm 1 Checkpoint ticker algorithm

```
1: Set checkpoint-ticker time and version-group labels when create the container
2: Swarm Manager annlyzes labels
3: while Container running & ticker time pounding do
4:     if version % version-group == 0  then
5:         pre-dump checkpoint image
6:     end if
7:     dump checkpoint image
8:     if  version-group directory > 3  then
9:         delete oldest pre-dump checkpoint directory
10:    end if
11:    version = version + 1
12:    if version % version-group == 0  then
13:        pre-dump = pre-dump + 1
14:    end if
15: end while
```

3.4.2 Docker Swarm restore rescheduling policy

Step 1. Set reschedule:restore label when we create the container.

Step 2. Swarm Manager analyzes restore label when the container start.

Step 3. Whenever Swarm Nodes fail, Swarm Manager will restore the containers which Swarm Manager has dumped the last checkpoint version to another Swarm nodes.

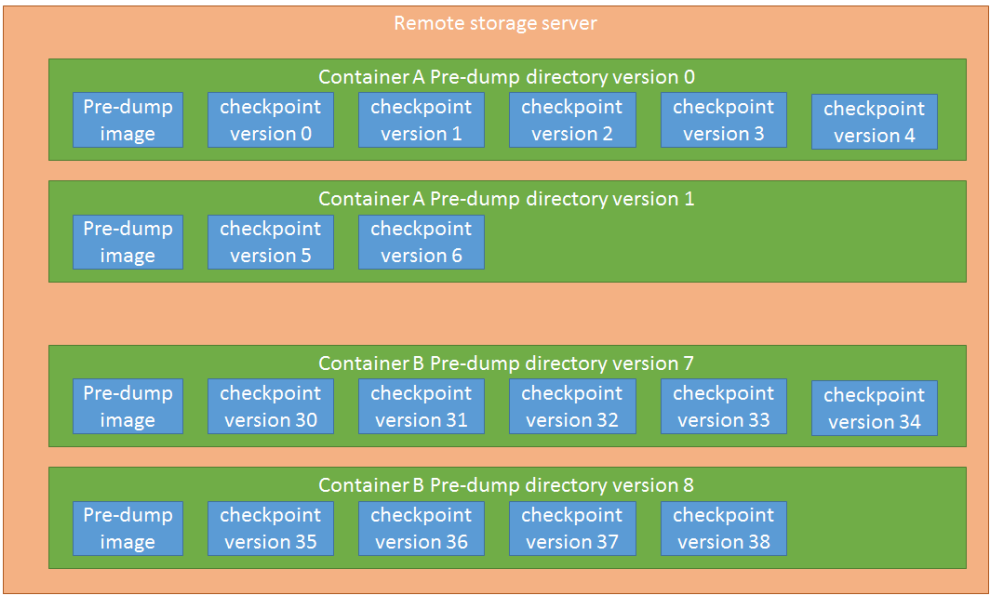


Figure 3.2: Containers checkpoint versions in remote storage server

Step 4. To avoid dumping checkpoint version failing at the same time, if restoring last version failing, Swarm Manager will retry second last version checkpoint to restore. It will retry version-group(default 5) times.

Step 5. If Swarm Manager retries version-group all fail, it will create and start a new container as normal Docker Swarm rescheduling policy.

3.4.3 High availability of Swarm Manager in Docker Swarm checkpoint and restoration rescheduling policy

Whenever Docker Swarm primary manager fails, the others Swarm Manager replica instances will lead a new primary manager. After replica leading a new primary manager, it searching every Docker Node's containers which has checkpoint restore rescheduling policy's label. If the containers has checkpoint restore rescheduling policy's label, Docker Swarm new primary manager will restart container checkpoint tickers.

Algorithm 2 Restore rescheduling policy algorithm

```
1: Set restore rescheduling policy labels create the container
2: Swarm Manager anlyzes labels
3:
4: if Swarm Node fail then
5:     Restore container
6: end if
7:
8: procedure Restore container
9:     for all containers on the fail Swarm Node do
10:        Create a empty container on the Docker Swarm Node.
11:        for version downto version - version-group do
12:            Restore checkpoint[version] container
13:        end for
14:        Delete the container checkpoint image
15:        if Restore the container fail then
16:            Restart the container on the Docker Swarm Node.
17:        end if
18:    end for
19: end procedure
```

Chapter 4

Experiments

ex

Chapter 5

Related Work

[4]

Chapter 6

Conclusion

CRIU provides the container for dumping checkpoint and restoring in the same Docker Daemon. In this thesis, we have purposed it to expand to Docker Swarm in the cluster.

Bibliography

[1] Criu. https://criu.org/Main_Page.

[2] Docker. https://www.docker.com/.

[3] Docker swarm. https://docs.docker.com/swarm/overview/.

[4] Michael R Hines and Kartik Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 51–60. ACM, 2009.