STRONG MIGRATION OF JAVA THREADS

SUMMARY

Today mobility is an important aspect of distributed applications and have several applications of use.For developing mobile applications, Java has been put forward as the reference platform. A Java program is compiled into machine-independent bytecode and can execute on any system, as long as a Java Virtual Machine(JVM) is installed on this system. Today, JVM is ported on almost every kind of hardware and software platform and can be seen as a universal machine and provides useful sevices for mobility of code and data.

But Java does not provide any service for serialization of the running applications. If a running application migrates to a new location, only by using object serialization and class loading mechanisms, the execution state of application is lost. It means, in the new location the migratory application can access its code and data but has to restart the execution from the beginning To migrate a running application, some state information has to be saved and transferred to the new location. In this new location, the application has to be restarted ideally in the exactly same state and at the same code position as it is before the migration.The main issue when building Java thread migration is to be able to access the thread's execution state, a state that is internal to the JVM and is not directly accessible to Java programmers.

There are someprojects have chosen to add new functionality to the Java enviroment in order to export the thread state from the Java Virtual Machine. This solution grants full access to the entire state of Java thread and in addition it is dependent to a specific extension of JVM which means this solution can not be used with the current Java virtual machines. In order to address Java thread migration, some projects use a solution on application level, without having a specific JVM restriction. In this approach, the application code is preprocessed before the execution in order to attach a backup object to the Java program executed by the thread, and to add new statements to this program. The new statements manage the execution state capture and restoration operation and store the state in backup object.

Since Java classes are compiled into transportable binary class files, the appropriate and platform-independent way to implement these improvements is not by developing a new compiler or changing the JVM, but by transforming the byte code. These transformations can be performed after compile-time, or at load-time. To deal with the necessary class file transformations, Byte Code Engineering Library(BCEL) API helps developers to implement their transformations properly.The Byte Code Engineering Library is designed to give users a possibility to analyse, create, and manipulate (binary) Java class files.

Brakes thread serialization technique is described in the context of middle ware support. It presents a simple execution model that guarantees correct thread migration semantics when moving a thread to another location.This thread serialization mechanism is implemented by extracting the state of a running thread from the application code that is running in that thread. To extract the state of a running thread, a **byte code transformer** modifies the application code by inserting code blocks that perform the actual capturing and reestablishing of the current thread's state. So this thread migration mechanism is portable across standard JVM platforms.Brakes offers tasks as a complement to JVM threads, in order to perform this thread serialization mechanism practical for migration. The JVM thread is encapsulated by a task, whose exectuion state is same with its threads execution state. Context object is defined for each task and the captured/reeastblished execution state is hold within this Context object.Transformer generates state capturing and state reestablishing code blocks in the application code.

But Brakes is not designed for migration between different JVMs.We will propose an extension to Brakes, named RemoteBrakes, to be able to move the threads between different JVM instances and allow context-switching between tasks within different JVMs.

By using Remote Method Invocation (RMI) of Java, an object running in one Java virtual machine to invoke methods on an object running in another Java virtual machine.

In this work, two different kind of applications are developed with RemoteBrakes&RMI integration. In this first application a distributed matrix calculation is demonstrated, one of the threads migrates in middle of its job and continues at the new JVM within a different task object. Second application is a Bouncing Balls application, one of the balls migrates to the other JVM with its task. With this application, migration is visual and you can track the path of the migrator ball.

The cost of Strong Migration of Java threads, is 11% to 20% space overhead on application codes.

JAVA İPLİKLERİNİN BAĞLAM BİLGİSİYLE BİRLİKTE GÖÇ ETTİRİLMESİ

ÖZET

Günümüzde mobilite/göç edebilirilik dağıtılmış uygulamalar için önemli bir konudur ve kullanılabileceği pek çok alan mevcuttur. Mobil uygulamalar geliştirmek için Java referans platform olmuştur. Bir Java programında makine bağımsız bytecode derlenir ve üzerinde Java Sanal Makinesi (JVM) kurulu olan tüm sistemlerde çalıştırılabilir. Bugün, JVM hemen hemen her tür donanım ve yazılım platformuna taşınmıştır, dolayısıyla evrensel bir makine olarak görülebilir. Vaynı zamanda kod ve veri taşınabilirliği için yararlı servisler sunmaktadır.

Fakat Java uygulamaları çalışma zamanında serileştirme için herhangi bir hizmet sağlamaz. Eğer çalışmakta olan bir uygulama, yalnızca nesne serileştirme ve sınıf yükleme mekanizmaları kullanarak yeni bir konuma göç ettirilirse, uygulamanın bağlam bilgisi kaybolur. Göçmen uygulamanın koduna ve verisine erişebilirsiniz ancak uygulamaya çalışmaya kaldığı yerden değil de baştan başlar. Çalışma zamanındaki bir uygulamayı göç ettirdiğimizde o zamana kadarki bağlamının korunmuş olmasını ve göç ettiği yerde kaldığı yerden çalışmasını bekleriz. Burada karşımıza çıkan sorun JVM'in bu yürütme bilgilerini programcılara vermemesi ve kendi içinde saklamasıdır.

Bazı mevcut projelerde Java ipliklerinin bağlam bilgilerine erişmek için JVM üzerinde değişiklik yapılmıştır. Böylelikle JVM'in normalde programcıya vermediği bilgilere erişilmiştir ancak Javanın platform bağımsızlığı ilkesi sekteye uğramıştır, çünkü bu bilgiye ulaşabilmek için değiştirilmiş JVM kullanmak zorunda kalınacaktır. Java ipliklerinin bağlam bilgilerine ulaşmak için diğer bir yöntem uygulama seviyesinde bu bilgilere ulaşmayı hedeflemiştir. Uygulama koduna eklenen bazı kod parçalarıyla ipliklerin bağlam bilgilerini saklanması ve gerektiğinde aktarıldığı yerde baştan yüklenmesi sağlanmıştır.

Java sınıflarının platform bağımsızlığı korumak için gerekli değişikliklerin bytecode seviyesinde yapılması en mantıklı çözümdür. ByteCode Mühendislik Kütüphanesi Uygulama Geliştirme Arayüzü(BCEL API) sayesinde bağlam bilgilerini alabilmek için gerekli değişiklikleri bytecode seviyesinde yapabiliriz. BCEL kullanıcılara bytecode analizi, baştan yaratmak ve değişiklik yapmak için gerekli tüm yapıları sunmaktadır.

Brakes İplik serileştirme mekanizmasıyla basit bir yürütme modeli kullanarak ipliklerin bağlam bilgisiyle göç ettirilmesi sağlanmaktadır. ByteCode Değiştirme mekanizmasıyla bağlam bilgisi saklayan ve yükleyen kodlar uygulamaya bytecode seviyesinde eklenir. Dolayısıyla Javanın platform bağımsızlığı korunmuş olur. Brakes ByteCode Değiştirme mekanizmasını yanı sıra her bir ipliği bir Görev (Task) ile ilişkilendirir ve görevin bağlamı ile ipliğin bağlamı aynı olacak şekilde bir yapı kurar. Görev içinde taskın bağlamının tutulduğu özel bir sınıf olan Bağlam bulunur. Bytecode seviyesinde eklenen kodlar bu Bağlam sınıfını güncelleyerek gerekli bağlam bilgisini tutmuş olur.

Ama Brakes farklı JVM'ler arasında göç için tasarlanmış değildir. Biz Brakes için, farklı JVM arasında göçü mümkün kılan ve bağlamları farklı görevler arasında taşımaya da izin veren RemoteBrakes adında bir eklenti önermekteyiz.
Bir nesnenin başka bir Java sanal makinede çalıştıran yöntemleri çağırmasına olanak veren Uzaktan Yöntem Invocation (RMI) kullanarak JVMler arası göçü sağlayacağız.

Bu çalışmada, iki farklı türde RemoteBrakes & RMI ile entegrasyon uygulaması geliştirilmiştir. İlk uygulamada Dağıtılmış bir matris hesaplama gerçeklenmiştir, ve ipliklerden bir tanesi işinin ortasında JVM 'e göç edip çalışmasına başka bir görev nesnesi içinde devam edebilmektedir. Ikinci uygulama Seken Toplar uygulamasıdır, burada da bir topun (ipliğin ve görevin) başka bir JVM'e göç etmesi ve çalışmasına kaldığı yerden devam etmesi simüle edilmiştir. Bu uygulamayla topların hareketleri görsel olarak takip edilebildiğinden demo amaçlı bir uygulamadır.

Bağlam bilgisiyle beraber göç edebilmenin maliyeti kodların boyutunda %11 ile %20 civarında bir artıştır.