分布式系统动态配置技术的研究与实现*)

窦 蕾[1] 王树凤[1] 徐田荣[2]

(国防科技大学计算机学院 长沙 410073)[1] (中国电子设备系统工程公司计算站 北京 100089)[2]

摘 要 动态配置技术用于系统运行时刻改变系统配置,从而满足分布式系统对在线演化的需求。如何保证系统一致性,并尽可能提高动态配置性能,是动态配置研究需要解决的关键问题。本文提出了行为一致性,并将系统一致性总结为行为一致性、构件状态一致性、应用状态一致性和引用一致性。通过扩展静止状态理论,本文提出了采用等待方式和阻塞方式驱动构件进入静止状态的算法,保证了行为一致性。通过分析动态配置意图,本文提出了保证系统一致性的动态配置算法,确定了采用等待方式和阻塞方式的动态配置场景,从而在严格保证行为一致性的前提下,尽可能地提高了动态配置性能。最后,本文基于CCM构件平台实现了动态配置平台。性能测试的结果表明,该平台可满足应用对性能的要求。

关键词 分布式系统,动态配置,系统一致性,CCM

Research and Implementation of Dynamic Reconfiguration Technology in Distributed System

DOU Lei[1] WANG Shu-Feng[1] XU Tian-Rong[2]

(School of Computer Science, National University of Defense Technology, Changsha 410073)[1]

(Computing Unit, China Electronic Equipment Systems Engineering Corporation, Beijing 100089)[2]

Abstract Dynamic reconfiguration aims to evolve distributed systems online by changing systems' configurations at runtime. How to preserve system consistency and to improve reconfiguration performance are the key issues in the research of dynamic reconfiguration. The paper presents behavioral consistency and summarizes the system consistency as: behavioral consistency, component state consistency, application state consistency and reference consistency. By extending the quiescent state theory, the paper proposes algorithms to drive component into quiescent state in waiting or holding mode so that the behavioral consistency is preserved. Based on the analysis of reconfiguration intentions and by choosing proper mode to drive component into quiescent state, the paper presents reconfiguration algorithms which improve reconfiguration performance as much as possible with the precondition that system consistency is preserved strictly. Finally, the paper implements the reconfiguration platform based on CCM, which is proven to meet the performance requirement of applications through the performance test.

Keywords Distributed system, Dynamic reconfiguration, System consistency, CCM

1 引言

从软件体系结构的角度出发[1],任何系统都可视为由构件根据一定的规则连接而成。根据系统的具体实现方式,抽象的构件概念可能对应着进程、对象或者真正的分布式构件,如EJB和CCM等。系统的构件组成、构件的物理分布及构件间的连接关系等构成了系统的配置信息。在系统运行时刻,对系统配置的更改即为动态配置。具有自适应、自管理能力的系统和长期运行的关键业务系统,如金融系统、国家通讯基础设施和军事指挥自动化系统等,必须利用动态配置技术实现组成构件和体系结构的在线演化,以迎合新技术的发展,满足客户需求的变化,适应复杂、不断变化的环境,或实现QoS管理。基本动态配置意图包括构件删除、构件添加、构件替换、构件迁移、连接删除、连接建立、连接重定向以及构件属性设置。复杂动态配置意图可由基本动态配置意图组合得到。通过分析应用系统当前的配置信息、动态配置意图及约束条件,动态配置平台生成动态配置算法并实施动态配置。

动态配置研究需要解决三个关键问题[2]。系统在动态配置期间和动态配置后必须处于正常运行状态。这种约束被称为系统一致性。如何保证系统一致性是动态配置研究的首要问题,也是动态配置算法设计和动态配置平台实现的基础。保证系统一致性的前提是动态配置意图本身的合法性。性能是动态配置研究的另一个关键问题。倘若系统由于动态配置的实施几近停止,则动态配置失去了意义,因此应尽量减少动态配置对系统性能的影响。如何描述应用系统和配置变化是动态配置研究的第三个关键问题。虽然采用形式化工具描述系统结构、语义、约束和配置变化并进行系统一致性等相关性质的形式化验证非常有意义,但这部分工作尚处于研究阶段,只能对特定系统进行形式化描述和分析,而且没有考虑动态配置性能问题。本文采用非形式化工具XML描述应用系统及其配置变化,重点关注系统一致性和动态配置性能两个问题。在假设动态配置意图合法的前提下,本文将通过合理设计并实现动态配置算法和通用的动态配置平台,严格保证系统一致性,同时尽可能提高动态配置性能。

分布式构件技术CCM(CORBA Component Model, CORBA构件模型)[3]为动态配置的实现提供了良好支持。

*)基金项目:自然科学基金(No. 90104020)。

首先，CCM 支持完整体系结构信息的获取。CCM 构件模型不仅定义了构件提供服务端口，而且定义了构件需求服务端口。基于对称的接口定义，组装描述文件描述了构件间的组装连接关系，从而提供了静态体系结构信息。容器作为构件的运行环境，将截获的所有到达构件的请求代理给构件，并对构件实施管理。因此我们可利用容器捕获体系结构变化事件，从而获得动态体系结构信息。其次，通过扩展容器，并利用 CORBA[4] 平台的优良反射和控制能力，如截获器和 Location_Forward 等机制，可以方便地实现对构件行为的监控。第三，通过属性文件和 configuration_complete 机制，CCM 直接支持对构件状态和应用状态的配置。最后，CCM 的组装部署模型，支持应用由构件组装而成，并被自动部署在分布式环境中。动态配置的本质就是应用在运行时刻重新组装并被部署的过程。CCM 的组装部署模型显然对其提供了最自然的支持。因此，我们决定基于 CCM 构件平台，利用其优良特性，并通过适当扩展，实现对分布式构件进行动态配置管理的动态配置平台。

本文第 2 节首先分析了系统一致性，将系统一致性定义为行为一致性、构件状态一致性、应用状态一致性和引用一致性，并提出了保证各种一致性的方法。本文第 3 节针对基本动态配置意图，设计了保证系统一致性的动态配置算法，分析了算法的正确性和性能。本文第 4 节介绍扩展 CCM 构件平台实现的动态配置平台原型系统，给出了性能评测。本文第 5 节介绍相关工作，最后进行总结并提出未来研究方向。

2 系统一致性

系统一致性刻画了系统的正确性，体现为动态配置在保证系统正确运行时所需满足的约束条件。应用系统在语义和实现方式上的差别，都会导致约束条件的不同。因此系统一致性的具体内容与应用系统密切相关。为了保证动态配置平台的通用性，并避免动态配置平台过于复杂从而导致性能、可靠性和可用性较低等问题，本文不致力于保证所有一致性，只是抽取出与具体应用语义和实现无关的公共一致性约束，并提出满足这些一致性约束同时兼顾性能的动态配置算法。现有动态配置研究抽取出的公共一致性包括以下 6 种[2,5,6]：

• 相互一致性（mutual consistency）。在系统运行过程中，构件间存在大量交互行为。如果在构件 A 响应构件 B 的请求时删除构件 A，将使构件 B 处于无法获得应答的异常状态，破坏了系统一致性。因此，动态配置必须保证构件间交互行为的完整性，这种约束被定义为相互一致性。

• 本地一致性（local consistency）。构件除了有交互行为外，还存在本地行为，如读写文件的行为。本地行为也对构件状态造成影响。对本地行为的随意中断，也可能引发系统异常，如打断读写文件操作所引发的系统 IO 异常。因此，动态配置必须保证关键的本地行为的完整性，这种约束被定义为本地一致性。

• 构件状态一致性（component state consistency）。在动态配置过程中，构件间传递状态的情况广泛存在，比如构件迁移、替换等。动态配置平台必须保证构件状态被正确收集，并可根据应用需求进行适当转换，从而正确初始化接收状态的构件，这就是构件状态一致性。

• 应用状态一致性（application state consistency）。应用系统可能存在系统断言，对系统中一组构件的状态进行了约定。例如，在基于令牌环结构的系统中，所有构件只能拥有一个令牌的约定。动态配置不能破坏这种全局的状态约束，这就是应用状态一致性。

• 引用一致性（reference consistency）。构件通过持有目标构件的引用，向其发送请求。当目标构件的引用由于动态配置的实施而被改变时（如构件迁移），动态配置平台应及时更新客户端持有的目标构件的引用信息，并保证所有基于旧构件引用发送的请求能够正确到达目标构件，此为引用一致性。

• 结构一致性（structural consistency）。根据特定语义，应用系统可能对系统结构存在特殊约定。例如，在基于令牌环结构的系统中，构件只能组合成环形结构。动态配置必须满足这种结构约束，此为结构一致性。

在上述 6 种一致性中，对结构一致性的保证主要依赖于对动态配置意图合法性的验证。在假设动态配置意图合法的前提下，本文无需考虑结构一致性。通过将相互一致性和本地一致性统一为行为一致性，本文保证的系统一致性包括：行为一致性、构件状态一致性、应用状态一致性和引用一致性。

2.1 行为一致性

系统在运行时刻存在大量正在进行的行为，包括构件交互行为和本地行为。动态配置必须保证系统行为的完整性，此为行为一致性（Behavioral Consistency，BC）。

相互一致性强调了交互行为的完整性，本地一致性强调了本地行为的完整性。若本地行为在执行过程中启动了若干交互行为，则本地行为的完整性将依赖于交互行为的完整性[5]。这种本地行为和交互行为之间的依赖关系实际强调了由若干本地行为和交互行为所构成的整体行为的完整性。事实上，无论是相互一致性还是本地一致性，都是要保证某个行为序列的完整性，这就是本文提出的行为一致性。这个行为序列可能由若干交互行为或本地行为单独组成，也可能由若干交互行为和本地行为混合组成，完全由应用根据其语义和实现决定。我们并不关心行为序列的具体组成，而将其直接定义为事务，并通过扩展静止状态理论[7]，简洁地保证了事务的完整性，进而保证了行为一致性。行为一致性的提出，不仅统一了相互一致性和本地一致性，而且对于简化保证行为序列完整性的算法设计也有很大帮助。

2.1.1 事务

构件为实现某意图启动一个事务。事务由若干构件的若干行为组成，包括构件交互行为和构件本地行为。事务代表了系统从一个一致性状态向另一个一致性状态迁移的过程。而在事务执行的过程中间系统状态是不一致的，因此事务中的行为序列不能被打断。动态配置必须保证系统中所有事务的完整性，从而保证行为一致性。

事务的执行过程可被抽象为一个自动机。所谓事务完整性，即指事务要不没有启动，一旦启动，则系统必须依据自动机的设计按照预定轨迹在状态间迁移，并最终到达预期的一致性终结状态。事务执行轨迹的改变，可能导致事务无法完成或进入异常状态。但在某些动态配置场景中，即使事务不能按照预定轨迹完成，系统也可以到达一个非预定的一致性状态，仍然满足系统的正确性要求，系统一致性不会被破坏。为了判断在何种情况下对事务执行轨迹进行何种改变不会破坏系统一致性，需要对系统行为和动态配置行为进行形式化的描述和分析。但是目前对系统形式化描述和分析的研究还远远没有达到实用程度，对动态配置的支持就更是不足。因而通用的动态配置平台尚无能力精确判断事务执行轨迹的改

变对系统一致性所造成的影响，只能通过严格要求事务按照设定轨迹执行，从而保证系统一致性。事务的完整性约束具体表现为事务原子性约束、事务语义合法性约束、事务保序性约束和事务封闭性约束。

定义 1 事务原子性约束 TAC(Transaction Atomicity Constraint)：事务的执行过程必须是原子的。事务要不没有启动，一旦启动，则组成事务的所有行为单元都必须完成。

定义 2 事务语义合法性约束 TVC(Transaction Valid Constraint)：事务中各行为单元完成的语义必须符合事务预期。

定义 3 事务保序性约束 TSC(Transaction Sequence Constraint)：事务中行为单元的串行执行关系不能被破坏。

定义 4 事务封闭性约束 TCC(Transaction Closure Constraint)：组成事务的行为序列中不能加入新的行为单元。

满足 TAC 是保证事务完整性的基本要求。事务一旦被启动，TAC、TVC、TSC 和 TCC 将共同保证系统按照预定轨迹运行，并达到预期的一致性终结状态。因此，行为一致性 BC 可被严格定义为：

$BC \equiv TAC \wedge TVC \wedge TSC \wedge TCC$

对于任意构件 A 而言，系统中的事务分为四类：构件 A 启动的事务；由其它构件启动但需要构件 A 参与的事务；构件 A 不直接参与但受其状态影响的事务；构件 A 不直接参与且不受其状态影响的事务。对构件 A 实施动态配置时，显然第四类事务的完整性不受影响。第三类事务与构件状态之间的关联关系，通常体现为系统对若干构件状态的全局约束。应用状态一致性将保证动态配置不会破坏这种全局的状态约束，从而保证第三类事务的完整性。因此，保证行为一致性时，只需考虑如何保证第一、二类事务的完整性。为此，本文扩展了静止状态理论，并将在相关工作中对扩展前后的静止状态理论进行比较。

2.1.2 静止状态理论

为保证行为一致性，构件在动态配置前必须进入静止状态。

定义 5 构件静止状态(quiescent state)被定义为：

1. 构件当前没有参与自行启动的事务；
2. 构件将来不会自行启动事务；
3. 构件当前没有参与由其它构件启动的事务；
4. 构件将来不会参与由其它构件启动的事务，包括已经启动和即将启动的需要该构件参与的事务，直到动态配置结束。

根据对构件静止状态的定义，构件启动和参与事务的行为被分为四类：

- B1：构件当前参与自行启动事务的行为；
- B2：构件将来自行启动事务的行为；
- B3：构件当前参与由其它构件启动的事务的行为；
- B4：构件将来参与由其它构件启动的事务的行为。

通过监控上述四类行为，可驱动构件进入静止状态。具体来说，驱动构件进入静止状态的方式有两种：等待方式和阻塞方式。两者的共同点在于，都是通过禁止 B2 类行为的发生并等待 B1、B3 类行为的完成，来满足静止状态定义的前三个条件。为满足第四个条件，等待方式在禁止其它构件启动需要目标构件参与的事务后，等待已经启动的此类事务的完成；阻塞方式则暂时阻塞到达构件的事务并延迟到动态配置结束后完成。对事务的阻塞体现为对事务中请求的阻塞。

采用等待方式驱动目标构件进入静止状态后，目标构件启动和参与的所有事务只可能处于两种状态，没有启动或者已经正常完成。此时对目标构件实施任何动态配置意图都不会破坏事务的完整性。因此，等待方式可在所有动态配置场景下严格保证行为一致性。

与之相比，阻塞方式只能保证特定动态配置场景下的行为一致性。采用阻塞方式驱动目标构件进入静止状态后，目标构件启动和参与的所有事务可能处于三种状态：没有启动、正常完成、暂被阻塞而尚未完成。为保证行为一致性，必须保证被阻塞事务的完整性。但在某些动态配置场景中，这个要求无法满足。实施构件删除意图时，被阻塞的到达目标构件的请求无法在动态配置结束后得到正常响应，违背了事务原子性约束。实施构件替换意图时，由于替换前后的构件可能具有不同实现，即使替换后的构件可继续响应被阻塞事务的后继请求，其响应请求的方式和结果可能也并不符合事务原有的预期，违背了事务语义合法性约束。严格说来，只有在实施构件迁移意图时，采用阻塞方式驱动被迁移的构件进入静止状态，可以保证行为一致性。因为作为构件替换的特例，构件迁移仅改变构件的物理位置，不改变构件的实现和接口等内容。而通常情况下，构件物理位置的改变不影响构件参与事务的语义。因而迁移后的构件可以按照预期完成被阻塞的事务，保证事务的完整性。阻塞方式虽然适用范围有限，但与等待方式相比可有效提高动态配置性能，因为阻塞方式无需等待事务完成，而且其实现过程对系统影响较小。

我们在设计动态配置算法时，将根据各动态配置意图特性，选择使用等待方式和阻塞方式驱动目标构件进入静止状态，从而达到在严格保证行为一致性的前提下尽可能提高动态配置性能的目标。下面我们分别介绍采用等待方式和阻塞方式驱动构件进入静止状态的方法。

等待方式。当构件满足静止状态定义中的条件 1、2 时，构件不存在自行启动的事务，只会被动参与由其它构件启动的事务，称其处于被动状态(passive state)。相对于被动状态，构件既可以参与由其它构件启动的事务，又可以自行启动事务的状态为主动状态(active state)。构件 Q 的被动集合(Passive Set)PS(Q)，由 Q 和所有可以启动需要 Q 参与的事务的构件组成。

在假设系统中所有事务都可在有限时间内独立完成的前提下，我们证明构件的静止状态和被动状态满足以下三条性质：

性质 1 构件的被动状态可达。

证明：根据前提假设，系统中的任何事务都在有限时间内独立完成。即使参与事务的构件处于被动状态，也不会阻止事务在有限时间内完成。因为处于被动状态的构件只是不具备启动事务的能力，但仍然具有正常参与事务的能力。因此构件自行启动的事务必定在有限时间内结束，静止状态定义中的条件 1 被满足。另外，只要构件能够在有限时间内禁止将来自行启动事务，就可在有限时间内满足静止状态定义中的条件 2。因此构件的被动状态可达。

性质 2 当 PS(Q)中的所有构件都进入被动状态时，构件 Q 进入静止状态。

证明：根据静止状态定义，Q 进入静止状态必须满足 4 个条件。当 Q 进入被动状态时，条件 1、2 被直接满足。当 PS(Q)中除 Q 以外的其它构件都进入被动状态时，表明当前和将来都不存在需要 Q 参与的事务，条件 3、4 得以满足。定理

成立。

性质 3 构件的静止状态可达。

证明:根据性质 1 和性质 2,性质 3 显然成立。

性质 2 实际给出了采用等待方式驱动构件 Q 进入静止状态的算法:首先计算 PS(Q),在驱动并确认 PS(Q)中的所有构件进入被动状态后,Q 进入静止状态。通过驱动 PS(Q)中除 Q 以外的所有构件进入被动状态,保证其它构件不再启动需要 Q 参与的事务,并等待已经启动的此类事务的完成,体现了等待方式的处理特点。

构件必须实现 passivateComp 方法,由执行动态配置算法的动态配置管理器(Dynamic Reconfiguration Manager, DRM)调用。构件在该方法中首先禁止将来自行启动事务,然后在确认所有自行启动的事务全部完成后,通知 DRM 自己已进入被动状态。为了在动态配置结束后使系统恢复到正常运行状态,构件还必须实现 activateComp 方法,在动态配置结束后被 DRM 调用,以驱动构件进入主动状态,恢复构件启动事务的能力。

动态配置的对象除了可以是单个构件外,还可以是由多个构件组成的构件模块。对于目标构件集合 TS(Target Set)来说,$PS(TS)=\bigcup_{i\in TS} PS(i)$。只有当 PS(TS)中的所有构件都进入被动状态时,模块中的各构件都进入静止状态,整个模块才进入静止状态。

阻塞方式。为驱动构件进入静止状态,阻塞方式和等待方式对 B1、B2 和 B3 类行为的处理方式是相同的。两者均通过驱动目标构件进入被动状态,禁止 B2 类行为发生后等待 B1 类行为的完成,从而满足静止状态定义的前两个条件。对于阻塞方式而言,在阻塞到达目标构件的请求后,通过等待 B3 类行为的完成,静止状态定义的后两个条件也被满足。

虽然采用阻塞方式驱动构件进入静止状态的思路很简单,但算法设计的难点在于对请求的正确阻塞。通常情况下必须对请求进行有选择的阻塞。原因在于,完全阻塞到达目标构件的请求,可能阻塞直接或间接来自其它目标构件的请求或重入请求,从而导致目标构件无法进入静止状态。对于任意目标构件 A,其接收的直接或间接来自其它目标构件的请求和重入请求可能属于以下两类事务:

- 目标构件启动的事务。目标构件 B 启动事务,事务中的请求 r 到达构件 A。当 A=B 时,r 为重入请求。
- 目标构件参与的由非目标构件启动的事务。非目标构件 C 启动事务,事务中的请求 r1 到达目标构件 B,B 为了响应 r1 直接或间接向构件 A 发送请求 r2。当 A=B 时,r2 为重入请求。

显然,上述两类事务不能被阻塞,否则可能阻塞目标构件发出的请求。我们将启动上述两类事务的构件组成的集合标记为 NBS(Not Blocking Set)。在驱动 NBS 中的所有构件进入被动状态后,上述两类事务都处于没有启动或已经完成的状态,因而任意目标构件都不会接收到直接或间接来自其它目标构件的请求以及重入请求。此时阻塞所有到达目标构件的请求,不会对目标构件进入静止状态造成任何影响。虽然并非所有目标构件启动的事务都符合第一类事务的要求,但考虑到所有目标构件都必须进入被动状态,因此本文扩展 NBS,将 TS 作为 NBS 的子集。启动第二类事务的构件构成了扩展集合 ES(Extended Set)。TS 作为目标构件的集合,本身就应进入被动状态。而驱动 ES 中构件进入被动状态的目的仅在于实现请求的正确阻塞。一旦开始阻塞所有目标构件,即使 ES 中的构件再启动事务,也不会影响目标构件进入静止状态。因为,由 ES 中构件启动的事务在到达第一个目标构件时就被阻塞,不会引发上述由第二类事务导致的目标构件发出的请求被阻塞的现象。因而,此时应立即将 ES 中的构件恢复到主动状态,尽可能减小动态配置对系统性能的影响。

综上所述,采用阻塞方式驱动目标构件进入静止状态的算法为:

1. 根据 TS 计算 ES,NBS=TS∪ES;
2. passivateComp(NBS);
3. block(TS);
4. activateComp(ES);
5. 在判定 TS 中的目标构件完成当前参与的事务后,目标构件进入静止状态。

采用等待方式或阻塞方式驱动构件进入静止状态时,精确计算 PS 和 ES 是提高动态配置性能的关键。由于 PS 和 ES 的确定与语义相关,而目前又缺乏描述和分析系统语义的有效手段,因此 PS 和 ES 尚不能被精确计算。我们在适度要求开发者提供一定语义信息的基础上,利用系统当前的结构信息,确定系统中的请求路径,从而动态计算出尽量接近真实集合的 PS 和 ES 的替代集合。

为获取完整的体系结构信息,从而正确计算 PS 和 ES,为对系统中的任意构件进行必要的控制,如驱动构件进入被动状态,静止状态理论正确实施的前提是:应用系统必须是全封闭、全可控的。但通常应用系统仅有一部分落在动态配置平台的控制边界内。如何基于有限的系统结构信息和控制能力,正确驱动控制边界内的构件进入静止状态,就是所谓的边界问题。我们提出了系统封闭化过程,将边界外的所有实体抽象为特殊的 Client 构件,从而得到一个全封闭、全可控的应用系统视图。动态配置平台将针对该视图,正确驱动控制边界内的构件进入静止状态,从而解决边界问题。

由于篇幅有限,我们在此就不对 PS 和 ES 集合的计算问题以及边界问题进行深入分析了。

2.2 构件状态一致性

动态配置时,需要在构件间传递状态的情况广泛存在。状态可能从正常工作的构件传递给新加入的冗余构件,从而实现容错或负载均衡。状态也可能从替换或迁移前的构件传递给替换或迁移后的构件。收集并提供状态的构件被称为状态传递的源构件。接受状态并用其初始化自身的构件被称为状态传递的目标构件。状态一致性强调的是,必须将源构件正确收集的状态传给目标构件,而目标构件通过直接继承该状态或者根据语义约束对状态进行转化后正常运行。

需要收集什么状态,是否需要状态转化以及如何转化等问题由应用语义决定。因此,每个构件必须实现 externalize 和 initialize 方法,分别负责状态的收集和初始化。动态配置管理平台仅负责在合适的时机调用 externalize 方法,获取收集的状态,并将其作为参数调用 initialize 方法,初始化构件。

通常,源构件在收集状态前必须通过等待方式进入静止状态,从而保证收集的状态是稳定的、一致的,不含有完成一半事务的状态。但是构件迁移时,由于迁移前后的构件完全相同,只要收集的状态是完整的,即使其中含有完成一半事务的状态,迁移后的构件也可基于这个初始状态完成迁移前构件未完成的事务,正常运行。因此构件迁移时,只需采用阻塞方式驱动源构件进入静止状态,即可开始状态收集。由于接

收状态的目标构件通常为新创建的构件，天然处于静止状态，因此为状态的初始化提供了理想的稳定环境。

2.3 应用状态一致性

构件状态的设定除了单纯与自身行为相关，还可能受到系统对状态全局约束的限制。应用状态一致性强调的是，在由于动态配置而导致全局状态约束被破坏时，必须重新设置构件状态，从而满足该约束。事实上，任何动态配置意图的实施都可能破坏全局的状态约束。

为保证应用状态一致性，设置构件状态的方式有两种：一种方式是，构件实现状态读写方法（如 get/set）。DRM 负责根据构件间的状态依赖关系，读写各构件状态，从而保证应用状态一致性。另一种方式是，DRM 在构件加入和离开系统时，调用构件的特定方法，由构件自行完成相关构件状态的设置，从而满足系统对状态的全局约束。与第二种方式相比，第一种方式可在任意时机给任意构件设置任意状态，更加灵活、有效。另外，在第一种方式中，构件仅提供通用的状态读写方法，不针对特定的系统状态约束嵌入特定的构件交互和状态设置代码，可方便地与不同构件组合成具有不同状态约束的应用系统。构件的重用性更强，应用范围也更加广泛。但是，采用第一种方式设置构件状态的前提是，构件状态必须可从外界明确指定。在复杂系统中，构件状态间的依赖关系比较复杂。某些状态产生于构件操作后，如创建构件操作产生的构件引用。这些状态很难从外界指定。对于这类状态的设定，采用第二种方式，由若干构件自行协商并设置构件状态则更加合理。两种方式各有优缺点，应结合使用。CCM 对这两种方式都提供了支持。对于第一种方式，CCM 支持通过读写构件属性的方式读写构件状态。对于第二种方式，CCM 支持利用 configuration_complete 机制在构件加入系统时设置构件状态。每个 CCM 构件都提供了 configuration_complete 方法。构件根据需要决定是否实现该方法。部署应用时，部署基础设施在创建完所有构件并建立构件间的连接后，将根据构件间的连接依赖关系，依次调用构件的 configuration_complete 方法，驱动各构件从部署阶段进入运行阶段。基于已建立起的连接关系，构件可在该方法中通过与其它构件交互设置若干构件的状态，从而满足系统对状态的全局约束。与 configuration_complete 对称的，我们扩展 CCM，引入 run_complete 机制。构件根据需要自行实现 run_complete 方法，用于在构件离开系统时设置系统中剩余构件的状态，从而保证应用状态一致性。

由于构件状态与构件行为相互影响，因此为了给构件状态的设置提供一个稳定的环境，同时保证行为一致性不被破坏，构件在属性设置方法或 configuration_complete/run_complete 方法被调用前必须通过等待方式进入静止状态。我们将这类为保证应用状态一致性而在状态设置前必须进入静止状态的构件称为系统状态设置构件。

2.4 引用一致性

构件持有的其它构件的引用可能来自名字服务、JNDI 等引用注册中心，也可能来自构件间的连接信息。例如，CCM 通过建立构件间的连接为构件设置其它构件的引用。构件通过查询连接信息获取目标构件的引用，并发送请求。对连接信息和注册的引用信息的更新并不能保证引用一致性，因为客户端构件可以利用先前获得的旧引用持续发送请求，而不进行重新查询，进而无法获得更新后的引用信息。何时更新引用信息，完全取决于客户端构件的实现。因此，为了保证引用一致性，在引用变化后，除了更新注册信息和连接信息外，还必须采用请求重定向机制将所有基于旧构件引用发送的请求重定向到正确的目标构件，而且请求重定向过程应对客户端构件完全透明。基于 CCM 的动态配置平台将利用 CORBA 的 Location_Forward 机制[4] 实现请求的透明重定向。

3 动态配置算法

针对构件删除、构件添加、构件替换、构件迁移、连接删除、连接建立、连接重定向以及构件属性设置 8 种基本动态配置意图，本文设计了 8 个动态配置算法。这些算法的主要实现步骤是相同的：根据特定动态配置意图和保证系统一致性的要求，确定需要进入静止状态的构件集合 QS（quiescent set）；驱动 QS 进入静止状态，从而使系统进入动态配置安全状态；执行动态配置操作，实现动态配置意图；驱动相应构件进入主动状态，恢复系统的正常运行。由于篇幅有限，本文仅以构件迁移意图为例，介绍动态配置算法的设计，分析算法的正确性和性能。

实现构件迁移意图时，由于旧目标构件的删除，可能中断事务的运行，因此需要保证行为一致性。新、旧目标构件间可能要进行状态的传递，因而需要保证构件状态一致性。新创建的目标构件具有与旧目标构件不同的引用，因而需要保证引用一致性。当系统存在对构件状态的全局约束时，还需要保证应用状态一致性。根据第 2 节对系统一致性的分析，为保证行为一致性和构件状态一致性，应采用阻塞方式驱动旧目标构件进入静止状态。为保证应用状态一致性，则应采用等待方式驱动系统状态设置构件进入静止状态。本文通过请求重定向和连接重定向保证引用一致性，具体参见构件迁移意图实现算法。若用 TS 标记旧目标构件集合，则实施构件迁移意图时，QS=TS∪{构件 n|n 为系统状态设置构件}。为使 QS 中的所有构件进入静止状态，根据 2.1.2 节给出的通过等待方式和阻塞方式驱动构件进入静止状态的算法，实施构件迁移意图前需要进入被动状态的构件集合 CPS（configuration Change Passive Set）为：CPS = PS(QS−TS) ∪ TS ∪ ES(TS)。其中 QS−TS 即为系统状态设置构件集合。构件迁移意图实现算法设计如下：

1）驱动 CPS 中的所有构件进入被动状态；

2）创建新目标构件；

3）阻塞到达新目标构件的所有请求；

4）连接重定向：将由其它构件启动的到达旧目标构件的所有连接原子性的重定向到新目标构件，并更新目标构件注册的引用信息；

5）请求重定向：将所有到达旧目标构件的请求重定向到新目标构件；

6）驱动 ES（TS）中的所有构件从被动状态进入主动状态；

7）判断并等待旧目标构件响应完毕所有请求；

8）构件状态传递：用旧目标构件收集的状态，初始化新目标构件；

9）调用旧目标构件的 run_complete 方法；

10）删除旧目标构件启动的连接；

11）删除旧目标构件；

12）建立新目标构件启动的连接；

13）为系统状态设置构件中通过属性方式设置状态的构件设置状态；

14)调用新目标构件的 configuration_complete 方法；

15)释放新目标构件处被阻塞的所有请求；

16)驱动 PS(QS—TS)中的所有构件从被动状态进入主动状态。

上述算法只是一个算法模板。根据配置者对动态配置意图的声明，动态配置平台将获悉哪些构件是需要迁移的目标构件、目标构件将迁移到哪里、是否需要进行构件状态的传递、是否需要保证应用状态一致性，哪些构件为系统状态设置构件，以及应为系统状态设置构件设置哪些状态等信息，从而自动生成针对特定构件迁移意图的、具体可执行的构件迁移算法。

连接重定向，由连接删除和连接创建过程组成，通过改变连接信息改变其它构件持有的目标构件的引用信息。在目标构件的迁移过程中，其它构件随时都可能获取连接信息并向目标构件发送请求。倘若构件在其与旧目标构件的连接被删除后而与新目标构件的连接尚未建立前试图获得连接信息，将引发异常，破坏行为一致性。因此，连接重定向强调的是必须保证由连接删除和连接建立构成的整个连接更新过程的原子性。连接重定向更新了旧目标构件注册在其它构件中的引用信息，请求重定向则将所有基于过时引用发送的请求重定向到新目标构件。两者共同作用，保证了引用一致性。

在基于 CCM 实现的配置平台中，构件除了接收到来自其它构件的请求外，还可能接收到来自 DRM 和部署基础设施的请求。我们将前者称为应用类型请求，将后者称为配置类型请求。在构件迁移意图实现算法中，无论是阻塞请求、重定向请求还是检测构件是否响应完毕所有请求都是指应用类型请求。配置类型请求在任何时候都被正常响应，从而保证各构件可在 DRM 的统一协调和控制下完成动态配置意图。

算法中的步骤 3 到步骤 5 完成了阻塞事务的 block(TS)过程。我们没有选择将请求先阻塞在旧目标构件处，最后再一起重定向到新目标构件的方法，而是将请求直接定向到新目标处进行阻塞。原因在于，使用 Location_Forward 机制实现请求重定向时，每个请求都要经过请求发送-获取重定向异常-请求再发送-获取应答的过程，显然大量请求的重定向将是一笔很大的性能开销。为了尽量减少请求重定向的发生，我们在阻塞事务时，尽早把请求正确定向到新目标构件。

由于连接重定向和请求重定向的存在，新目标构件响应请求的顺序与请求被发出的顺序并不相符，但这并不会破坏事务保序性约束。倘若事务中的构件交互行为采用同步通讯方式实现，即使请求被重定向，后继的请求也只能在前一个请求被成功响应后才能启动，因此事务中行为序列的执行顺序没有被破坏。倘若事务中的构件交互行为采用异步通讯方式实现，则即使在系统正常运行期间，中间件也不承诺先发的请求必定先被响应，因此在这种情况下破坏行为序列的执行顺序是被事务语义所允许的。综上所述，对于事务中需要严格保证串行执行顺序的动作序列而言，其实现方式必定是同步通讯方式，因而构件迁移算法不会破坏事务保序性约束。

根据对动态配置意图的分析以及构件迁移算法的设计，易证构件迁移算法保证了系统一致性。

4 系统实现与性能测试

我们基于前期遵循 CCM 规范实现的 StarCCM 构件平台[17]实现了动态配置平台。基于 CCM 的动态配置平台的体系结构如图 1 所示。

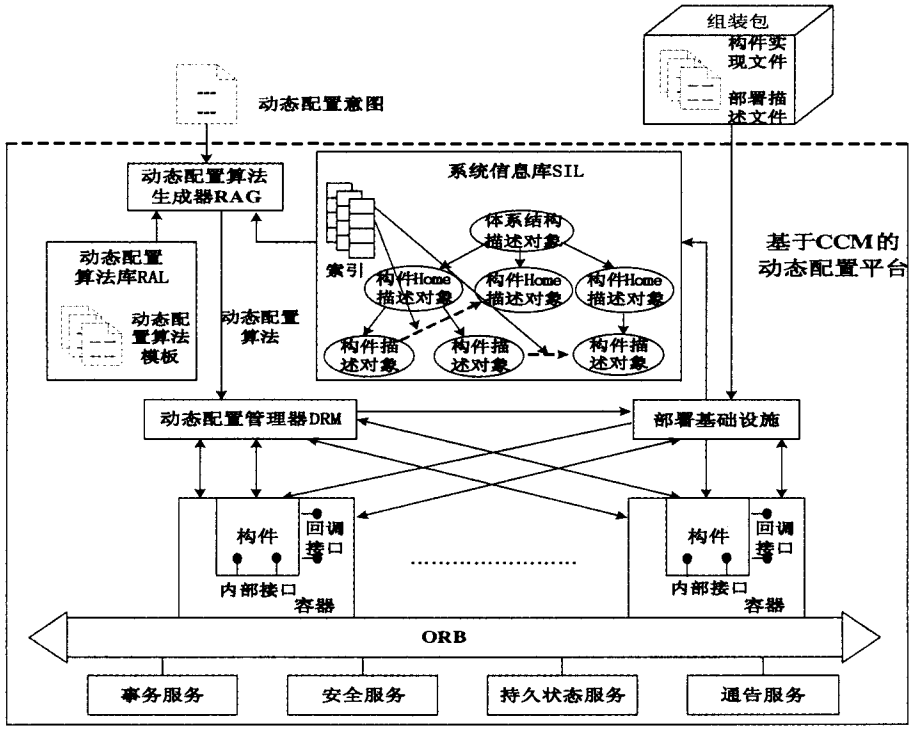

图 1 动态配置平台体系结构

动态配置平台由系统信息库(System Information Library,SIL)、动态配置算法库(Reconfiguration Algorithm Library,RAL)、动态配置算法生成器(Reconfiguration Algorithm Generator,RAG)、动态配置管理器(Dynamic Reconfiguration Manager,DRM)、容器和部署基础设施 6 部分组成。

系统信息库 SIL 中存放了系统结构和语义信息。部署基础设施在部署代表应用的组装包时，解析部署描述文件从而收集应用的组成构件、构件的物理位置、构件间的连接关系等系统静态体系结构信息以及各构件的语义信息，并将这些信息存放于 SIL 中。容器在应用运行过程中，捕获体系结构变

化事件，收集系统的动态体系结构信息。各容器收集的动态体系结构信息也将通过部署基础设施汇总存储于 SIL 中。为了支持系统信息在 SIL 中的合理组织，我们为 SIL 设计了层次化的系统信息存储结构。为了提高查询效率，我们还在 SIL 中建立了有关构件连接等信息的索引。基于对动态配置意图的分析而设计的动态配置算法作为模板存储于 RAL 中。根据动态配置者通过 GUI 描述的动态配置意图，RAG 从 RAL 中取出对应的动态配置算法模板，利用 SIL 中的系统信息，生成针对此次动态配置意图的动态配置算法，提交给 DRM。DRM 解析动态配置算法，通过与构件、容器和部署基础设施的交互，利用 CORBA 提供的请求截获和重定向等功能，进行构件的创建和删除、连接的创建和删除、请求的阻塞、释放和重定向、构件状态的收集、设置和检测等行为，最终实现动态配置算法，完成动态配置意图。基于 CCM 的动态配置平台提供的功能可被总结为 5 类动态配置基本机制：系统信息描述及计算机制、构件状态检测机制、构件行为控制机制、构件状态传递机制、动态配置算法描述机制。每种机制都由若干子机制组成，例如系统信息描述及计算机制包括系统信息收集机制、系统信息存储\查询机制以及系统信息计算机制。

我们基于 CCM 实现了一个负载均衡系统，并利用该系统对本文设计的动态配置算法进行了全面测试。测试结果表明，动态配置保证了系统一致性，从而验证了动态配置算法和动态配置平台的功能正确性。由于篇幅原因，本文不对该测试进行详细说明。

动态配置平台通过提供 5 类动态配置基本机制，为动态配置算法的实现提供了支持。但这些机制在系统正常运行时也对系统性能造成影响。因此动态配置平台的性能问题应从两方面考虑：一方面在系统正常运行时，要尽量减少各种机制对系统的影响；另一方面在动态配置时，要尽量提高动态配置实施的性能。

对于后者而言，合理设计动态配置算法，是提高性能的关键。本文已在系统一致性分析和动态配置算法设计中对此进行了分析。我们在设计、实现动态配置平台时将通过提高动态配置算法执行的并行度，进一步提高这方面的性能。我们没有针对动态配置实施时的性能展开具体的案例测试，原因在于动态配置的实施过程和对系统的影响程度与应用系统的语义和实现方式关系密切。单纯就任何一个特定的应用系统测试动态配置实施的性能，不能说明任何问题，意义不大。

通过分析动态配置平台的实现，我们发现动态配置平台对系统正常运行时性能的影响主要来源于构件状态检测机制和构件行为控制机制。为监控构件，我们利用 CORBA 的截获器机制、容器以及 CCM 构件与容器间的交互接口，在请求的正常处理路径上加入了若干控制点，从而实现对事务边界的记录、对请求类型的标识、对构件响应请求状态信息的记录等。针对这点，我们设计的测试案例主要测试动态配置平台对请求平均响应时间的影响，从而度量动态配置平台对系统正常运行时性能的影响。测试系统基于 ORBacus4.1.0 实现，由一个客户端构件 C 和一个服务端构件 S 组成。C 和 S 被部署在不同的主机上。每启动一次测试，C 就启动一个线程向 S 发送 10000 个请求，并在测试结束时统计 10000 个请求的平均响应时间 R。S 在响应请求时不做任何操作。$R=T_{middleware}+T_{network}+T_{application}$。其中 $T_{middleware}$ 代表请求和应答经过中间件的开销，动态配置平台引入的性能开销在此体现。$T_{network}$ 代表请求和应答在网络中传输的通讯开销。$T_{application}$ 代表服务端构件响应请求的开销，在我们设计的测试系统中，$T_{application}$ 为 0。因此 R 反映出来的就是中间件和网络的开销。我们在表 1 中给出了 3 组对比测试的结果，并在图 2 中对各组测试的平均响应时间的增长率进行了比较。第一组测试了在没加入动态配置功能的纯 CCM 平台上的平均响应时间 $R_{PlainCCM}$。第二组测试了在纯 CCM 平台上只加入截获器框架，而在截获点处不实现任何操作的平均响应时间 $R_{PlainPI}$。第三组则测试了在完整的基于 CCM 的动态配置平台上的平均响应时间 $R_{Reconfiguration}$。在每组测试中，我们还分别针对不同大小的请求参数和网络带宽进行了测试。其中各组测试的平均响应时间的增长时间和增长率的参照值都是 $R_{PlainCCM}$。

表 1　平均响应时间对比测试

带宽	参数大小	$R_{PlainCCM}$ (ms)	$R_{PlainPI}$ (ms)		$R_{Reconfiguration}$ (ms)	
			增长时间	增长率	增长时间	增长率
100M	0B	0.295947	0.052289	18%	0.133783	45%
10M	0B	0.477364	0.059104	12%	0.136069	29%
10M	128B	1.460699	0.065993	4.5%	0.149065	10%
10M	1 kB	8.727146	0.058906	0.7%	0.181287	2.1%
10M	2 kB	17.785531	0.089959	0.5%	0.297466	1.7%

随着网络带宽的减小和请求参数的增加，我们发现平均响应时间增长的绝对时间虽略有所增加，但是增长率却明显下降。原因在于，随着网络负载的加重，$T_{network}$ 在 R 中占的比例越来越高。相对于较大的通讯开销而言，动态配置平台所带来的性能开销，对系统性能的影响程度较小。我们测试的环境是一个网络相对轻载的环境：部署了客户端构件和服务端构件的两台机器通过 Hub 连在一起，构成测试的局域网，并与外界隔离。但在真实环境中，若干台机器将共享并竞争通讯带宽，通讯开销进一步增加，动态配置平台对性能的影响也必然更小，可以满足应用对性能的要求。通过测试，我们发现 CORBA 的截获器框架对性能影响较大。如果能通过改造 ORB 内核，有效降低截获器框架引入的性能开销，则动态配置平台对系统性能的影响将进一步减小。

5　相关工作

本文扩展的静止理论来自于 Jeff 方法[7]。在原静止状态理论中，事务仅包括构件交互行为，只能通过等待方式驱动构件进入静止状态，从而保证事务的完整性。原静止状态理论仅保证了相互一致性，而无法保证本地一致性。而且，原静止状态理论虽然定义了 PS，但没有给出 PS 计算算法，也没有对边界问题展开分析。本文在扩展静止状态理论时，引入了通过阻塞方式驱动构件进入静止状态的算法，分析并解决了 PS 计算问题和边界问题，进一步完善了静止状态理论，对提高动态配置性能以及确定静止状态理论的有效性边界都有重要意义。Warren 方法[5]虽然提出了本地一致性，但没有给出保证本地一致性的具体算法。Warren 方法将构件交互行为和本

地行为的完整性问题分开考虑，导致在保证由若干构件交互行为和本地行为所构成的整体行为的完整性时，实现方法十分复杂。本文提出的行为一致性，不仅统一了相互一致性和本地一致性，而且通过扩展静止状态理论简洁地保证了事务完整性，从而简化了保证构件交互行为和本地行为完整性的处理。

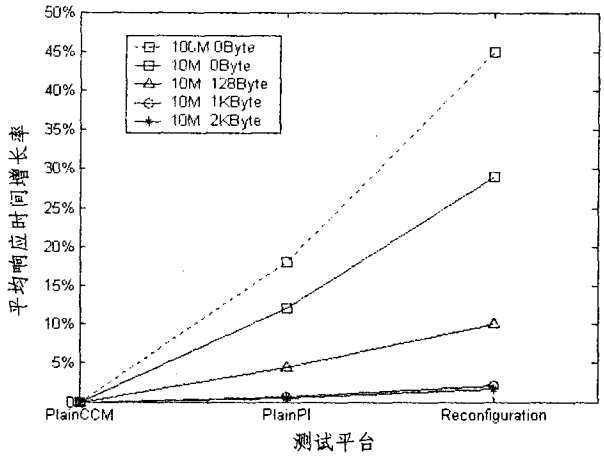

图 2　平均响应时间增长率对比图

为提高动态配置性能，也有很多方法采用阻塞方式驱动构件进入静止状态。Bidan 方法[8]和 XRMI 方法[9]没有考虑有选择的请求阻塞问题。Almeida 方法[10]利用 CORBA 的截获器机制记录请求路径信息，请求路径中不含目标构件的请求将被阻塞。这种方法用于记录和传输请求路径信息的性能开销较大，而且对系统正常运行时的性能影响很大。在 Goudarzi 方法[3]中，所有目标构件组成了初始阻塞构件集合 BSet。随着系统的运行，BSet 被动态扩展以加入直接或间接响应 BSet 成员请求的非 BSet 成员构件。阻塞请求时，来自 BSet 成员的请求将被正常响应，由此实现了请求的有选择阻塞。Goudarzi 方法对系统正常运行时的性能影响较小，但实现比较复杂，而且 BSet 动态扩展所带来的通讯开销，影响了动态配置时的性能。Wermelinger 方法[11]依照构件间的依赖关系顺序阻塞构件，保证目标构件在被阻塞时不会接收到直接或间接发自其它目标构件的请求，但前提是构件的依赖关系不构成环，因而该方法也无法正确处理重入请求。与上述方法相比，本文设计的有选择请求阻塞方法对系统正常运行时的性能没有任何影响，实现简单，动态配置性能较高，并可保证到达目标构件的直接或间接来自其它目标构件的请求以及重入请求都被正常响应。

基于容错 CORBA 平台[12]，OMG 制定了 CORBA 对象在线升级规范[13]，Tewksbury 方法则基于特定容错 CORBA 平台 Eternal 实现了 CORBA 对象的在线升级[14]。利用容错 CORBA 平台的特性，上述方法很自然地通过对象组引用机制解决了引用一致性，特别是提供了保证在线升级过程原子性的事务机制，支持当在线升级过程失败时对系统的恢复。但是这类方法基于容错 CORBA 平台实现，对应用系统提出了额外要求。而且两种方法仅保证了相互一致性，没有保证本地一致性，在采用阻塞方式驱动构件进入静止状态时，也没有考虑有选择的请求阻塞问题。

Fabio 方法[15]和 Minsky 方法[16]不针对任何特定的一致性约束，提出了通用的动态配置框架。Fabio 方法的核心是配置对象(Configurator)。系统中的每个构件都被一个配置对象管理。配置对象记录构件间的依赖关系，并负责在相互关联的构件间传播事件，以及处理接收到的事件，如某构件被卸载的事件。开发者可以自定义事件，并编写事件处理代码。基于应用的语义约束，Minsky 方法支持在每个构件的控制器中设定构件交互规则，从而通过控制构件的交互行为，实现对动态配置的支持。通用的动态配置框架允许开发者针对特定的应用语义、动态配置功能和对系统一致性的要求定制动态配置方法，提高了动态配置的性能，提供了灵活性和适应性。但是，配置对象的实现和交互规则的设定，都加重了开发者的负担，而且人的过多参与也降低了系统的可靠性。两种方法虽然从框架的角度来说可支持任何动态配置行为的实现，但对于特定应用来说，配置对象或控制器一旦实现，系统支持的动态配置功能和保证的系统一致性也就确定，并不能提供真正的自适应性。另外，两种方法都是通过构件自治的方式实现动态配置，系统无法从全局角度对动态配置进行必要的管理和优化。

总结现有动态配置方法中存在的不足，主要包括：

• 对系统一致性支持不全面。目前没有一种动态配置方法为已抽取出的 5 类公共一致性提供全面支持。

• 本地一致性的忽略以及与相互一致性的分离。大部分方法没有考虑本地一致性。Warren 方法虽然提出了本地一致性，但由于将构件交互行为和本地行为的完整性问题分开考虑，因而没有提出可行算法。

• 保证系统一致性和提高动态配置性能的对立。Jeff 方法采用等待方式驱动构件进入静止状态，严格保证了系统一致性，但性能较低。而采用阻塞方式的动态配置方法虽然提高了动态配置性能，却没有考虑当被阻塞的事务在动态配置结束后无法完成而导致相互一致性被破坏的情况。

• 动态配置平台实现复杂度较大。动态配置平台必须具备对构件行为的监控能力。但是传统的基于对象的中间件平台，如 CORBA 和 RMI 等，没有为上述机制提供良好支持，各动态配置方法需要对原有分布式平台进行较大扩展，如引入工厂对象和各种形式的动态配置代理对象，以实现对构件必要的监控，复杂度较大。

针对现有动态配置方法中存在的上述不足，本文为 5 类公共一致性提供了全面支持，并将相互一致性和本地一致性统一为行为一致性，还通过扩展静止状态理论，保证了行为一致性。本文又在静止状态理论中引入了阻塞方式，并通过分析各动态配置意图，确定了采用等待方式和阻塞方式的动态配置场景，从而在严格保证行为一致性的前提下，尽可能提高动态配置性能。最终，本文利用 CCM 构件平台为动态配置提供了良好支持，通过适当扩展，实现了基于 CCM 的动态配置平台，有效降低了动态配置平台的实现复杂度。

结束语　本文分析了动态配置过程中必须保证的系统一致性，并针对 8 种基本动态配置意图设计了保证系统一致性的动态配置算法。通过扩展 CCM 构件平台，本文设计并实现了基于 CCM 的动态配置平台。功能测试验证了该平台的正确性，性能测试则表明该平台可满足应用对性能的要求。

在下一步的工作中，我们将针对事务机制和基于动态配置的 QoS 管理进行深入研究。为了保证动态配置过程的原子性，从而在动态配置因种种原因而中途失败时可将系统恢复到一致性状态，动态配置平台应提供实现失效恢复的事务机制。鉴于容错 CORBA 平台在这方面提供了很好的支持，

(下转第 57 页)

疫来判别自我，可以缓解误报问题。在应用多层联合检测时还需要上层与下层间进行协调[9]。

(3)IDS系统自身的安全

IDS系统间通信应该保密和节点间进行认证，防止伪造IDS命令或它们之间的通信。而目前Ad Hoc网络的大部分研究中，对IDS的安全只是提及[9]，或者放在下一步工作中[20]。另外，由于Ad Hoc网络的特殊性，网络节点也容易被俘获，因此密钥等设施也可能被敌对者得知。可以引入信任，如Marti在文[10]中提到的使用先验信任(Apriori trust)，不过这种静态给定的信任度显然不能适应Ad Hoc网络的环境，可以采用动态改变信任度的方法。

结束语 Ad Hoc网络中的安全十分重要，已经出现了很多安全设施，诸如安全路由、加密、入侵检测等。其中入侵检测是最近才提出的，但由于其在安全中的重要性，并且由于Ad Hoc网络的特性而与传统有线网络存在很多不同，得到较多的关注。本文对当前Ad Hoc网络的入侵检测研究做了分析比较，指出了一些待解决的问题，在下一步工作中解决。

参考文献

1 Brutch P, Ko C. Challenges in Intrusion Detection for Wireless Ad-hoc Networks. In: 2003 Symposium on Applications and the Internet Workshops (SAINT'03 Workshops), 2003
2 Kachirski O, Guha R. Effective Intrusion Detection Using Multiple Sensors in Wireless Ad Hoc Networks. In: Proc. of the 36th Hawaii Intl. Conf. on System Sciences IEEE (HICSS'03), 2002
3 Hu Y C, Johnson D, Perrig A. SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks. In: Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '02), 2002. 3～13
4 Sun B, Wu K, Pooch W, et al. Alert Aggregation in Mobile Ad Hoc Networks. In: WiSE'03. San Diego, California, 2003
5 Huang Yi-an, Lee W. A Cooperative Intrusion Detection System for Ad Hoc Networks. In: 2003 ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '03), 2003
6 Albers P, Camp O, Percher J-M, et al. Security in Ad Hoc Networks: a General Intrusion Detection Architecture Enhancing Trust Based Approaches. In: The 1st Intl. Workshop on Wireless Information Systems, Proc. of the 4th International Conf. on Enterprise Information Systems, 2002
7 Buttyan L, Hubaux J P. Report on a Working Session on Security in Wireless Ad Hoc Networks Mobile Computing and Communications Review, 2002, 6(4)
8 Anjum F, Subhadrabandhu D, Sarkar S. Signature based Intrusion Detection for Wireless Ad-Hoc Networks: A Comparative study of various routing protocols. In: Proc. of Vehicular Technology Conference, Wireless Security Symposium, Orlando, Florida, Oct. 2003
9 Zhang Y, Lee W. Intrusion detection in wireless ad hoc networks. In: Proc. of the sixth annual Intl. conf. on Mobile computing and networking, MOBICOM'2000, ACM Press New York, USA, 2000. 275～283
10 Marti S, Giuli T J, Lai K, et al. Mitigating routing misbehavior in mobile ad hoc networks. In: Proc. of the Sixth Annual Intl. Conf. on Mobile Computing and Networking, 2000. 255～265
11 Lippmann R, Fried D, Graf I, et al. Evaluating intrusion detection systems: The 1998 darpa online intrusion detection evaluation. In: Proceedings of the 2000 DARPA Information Survivability Conference and Exposition, January 2000
12 Chlamtac I, Conti M, Liu J. Mobile Ad Hoc Networking: Imperatives and Challenges. Ad-Hoc Networks Journal, 2003, 1(1)
13 Ilgun K, Kemmerer R A, Porras P A. Statetransition analysis: A rule-based intrusion detection approach. IEEE Transactions on Software Engineering, 1995, 21(3): 181～199
14 Zhang Y G, Lee W, Huang Yi-An. Intrusion detection techniques for mobile wireless networks. Wireless Networks, 2003
15 Sarafijanovic S, Le Boudec J Y. An Artificial Immune System for Misbehavior Detection in Mobile Ad-Hoc Networks with Virtual Thymus, Clustering, Danger Signal and Memory Detectors. In: Proceedings of ICARIS-2004, 3rd International Conference on Artificial Immune Systems, Catania, Italy, September 2004. 342～356
16 Hu Y, Perrig A, Johnson D B. Ariadne: A secure on-demand routing protocol for ad hoc networks. In: Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking (MobiCom 2002), 2002
17 Zapata M G. Secure ad hoc on-demand distance vector (SAODV) routing. IETF Internet Draft, draft-guerrero-manet-saodv-00.txt, August 2001 (Work in Progress)
18 Krishna P, Vaidya N H, Chatterjee M, et al. A cluster-based approach for routing in dynamic networks. ACM SIGCOMM Computer Communication Review, 1997, 27(2): 49～64
19 Joa-Ng M, Lu I. A Peer-to-Peer zone-based two-level link state routing for mobile Ad Hoc Networks. IEEE Journal on Selected Areas in Communications, 1999, 17(8): 1415～1425
20 Vigna G, Gwalani S, Srinivasan K, et al. An Intrusion Detection Tool for AODV-based Ad hoc Wireless Networks. In: Proceedings of the Annual Annual Computer Security Applications Conference (ACSAC) Tucson, AZ December 2004
21 Deng H, Zeng Q-A, Agrawal D P. SVM-based Intrusion Detection System for Wireless Ad Hoc Networks. In: Proceedings of the IEEE Vehicular Technology Conference (VTC'03), 2003
22 Zhou L, Haas Z J. Securing ad hoc networks. IEEE Network Magazine, 1999, 13(6): 24～30

(上接第15页)

可考虑结合容错CORBA平台，或借鉴其失效恢复机制在动态配置平台中提供事务机制。动态配置技术为灵活高效的QoS管理提供了支持，包括容错和负载均衡等。如何将动态配置技术和容错、负载均衡等技术具体结合起来，实现系统的自管理和自适应，也有待研究。

参考文献

1 Shaw M, Garlan D. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996
2 Moazami-Goudarzi K. Consistency preserving dynamic reconfiguration of distributed systems: [Ph. D. thesis]. London: Imperial College, March 1999
3 Object Management Group. CORBA Component Model Specification V3.0. Formal/2002-06-65
4 Object Management Group. Common Object Request Broker Architecture: Core Specification. V3.0, formal/02-11-03, November 2002
5 Warren, Sommerville I. A model for dynamic configuration which preserves application integrity. In 3rd International Conference on Configurable Distributed Systems, 1996. 81～88
6 Hofmeister C R. Dynamic Reconfiguration of Distributed Applications: [Ph. D. Thesis]. University of Maryland, 1993
7 Kramer J, Magee J. The evolving philosophers' problem: dynamic change management. IEEE Transactions on Software Engineering, 1990, 16(11): 1293～1306
8 Bidan C, Issarny V, Saridakis T, et al. A dynamic reconfiguration service for CORBA. In: Proc IEEE International Conference on Configurable Distributed Systems, May 1998
9 Chen Xuejun. Extending RMI to Support Dynamic Reconfiguration of Distributed Systems. In: Proceeding of the 22nd International Conference on Distributed Computing Systems (ICDCS 2002)
10 Almeida J P A, van Sinderen W M, Nieuwenhuis L. Transparent Dynamic Reconfiguration for CORBA. In: Proc. of the 3rd International Symposium on Distributed Objects & Applications (DOA 2001), 2001. 17～20
11 Wermelinger W A. Specification of software architecture reconfiguration: [Ph. D. thesis]. Universidade Nova de Lisboa, September 1999
12 Object Management Group. Fault Tolerant CORBA Specification. V1.0, ptc/00-04-04, April 2000
13 Object Management Group. Online Upgrades. Draft Adopted Specification. OMG Document ptc/2002-07-01
14 Tewksbury L A, Moser L E, Melliar-Smith P M. Live upgrades of CORBA applications using object replication. In: Proc. IEEE International Conference on Software Maintenance, 2001. 488～497
15 Kon F. Automatic Configuration of Component-Based Distributed Systems: [PhD Thesis]. Department of Computer Science, University of Illinois at Urbana-Champaign, 2000
16 Minsky N H, Ungureanu V, Wang Wenhui, et al. Building Reconfiguration Primitives into the Law of a System. In: Proc. of the 3rd International Conference on Configurable Distributed Systems, 1996. 89～97
17 homepage of StarCCM on SourceForge website: http://sourceforge.net/projects/starccm/