



SaaS 支撑框架及关键技术研究^{*}

张小东^{1,2}, 战德臣¹, 聂兰顺¹, 孟凡超²

(1. 哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001;

2. 哈尔滨工业大学(威海)计算机科学与技术学院 威海 264209)

摘要:当前主流的 PaaS 和 IaaS 对于 SaaS 的支持存在一些问题:个性化定制因缺少语义关联而无法深入进行;以虚拟机或整体应用为单位的粗粒度管理在扩展时容易造成资源浪费;缺少对第三方 SaaS 在多租户方面的支持。为此,以 SaaS 构件为管理单位提出了 BIRISF,提高了管理的精细度,以分散热点方式进行扩展,减少了资源的浪费,通过建立领域模型与知识库支持 SaaS 的深度定制,并采用散列一致性反馈算法和三阶段动态扩展策略解决 SaaS 在扩展与性能方面的问题。经过在国际海运物流领域的应用,验证了 BIRISF 在开发、运维及性能方面均优于经典的多层架构。

关键词: BIRISF; SaaS 型构件; 散列一致性动态反馈算法

doi: 10.3969/j.issn.1000-0801.2013.12.012

Research on SaaS Support Architecture and Key Technologies

Zhang Xiaodong^{1,2}, Zhan Dechen¹, Nie Lanshun¹, Meng Fanchao²

(1. School of Computer Science and Technology, Harbin Institute of Technology (HIT), Harbin 150001, China;

2. School of Computer Science and Technology, Harbin Institute of Technology at Weihai, Weihai 264209, China)

Abstract: There are some problems about SaaS supported by the popular PaaS and IaaS: customizing SaaS is commonly not done deeply without semantic association; sources are wasted when SaaS applications are scaled by VM and whole application unit; SaaS applications developed by the 3rd party software vendors are not enough supported in multi-tenancy. So BIRISF was presented. Management was improved finely in way of managing SaaS components. SaaS applications were extended and waste of sources was reduced by spreading hot components on different nodes. An oriented-domain model and knowledge database were built to support to customize SaaS deeply. The problems of performance and scalability by consistent hashing feedback algorithm and three stages of dynamical extending and deploying strategy were solved. It was proved that BIRISF was superior to the classical multi-tier architecture in development, operations, maintenance and performance because of BIRISF applied successful in international shipping logistics.

Key words: bilateral resource integration service framework, SaaS component, consistent hashing feedback algorithm

1 引言

云计算有 SaaS (software as a service, 软件即服务)、

PaaS (platform as a service, 平台即服务) 和 IaaS (infrastructure as a service, 基础设施即服务) 三大服务模式^[1]。SaaS 是一种将软件及其相关的数据集中托管于云上

^{*} 国家科技支撑计划基金资助项目 (No.2012BAF12B16), 国家自然科学基金资助项目 (No.61273038), 山东省科技攻关项目 (No.2010GX20126, No.2010GGX10104, No.2010GGX10116), 威海—哈工大共建项目 (No.2011DXZJ07)

的软件运营模式,降低了软件系统的运维成本,获得了广泛的认可和使用^[2]。PaaS和IaaS源于SaaS理念,直接通过SOA/Web Services提供服务或作为SaaS应用的支撑平台间接提供服务。由此可知,SaaS处于云计算应用的核心位置,拥有可定制、可扩展、支持多租户(multi-tenancy,MT)等特性,可以满足租户在展示、功能、性能等方面的多样性需求,这些特性的实现需要位于云计算栈下层的PaaS与IaaS的支撑^[3]。然而,面对丰富的领域应用,当前主流的PaaS和IaaS对于SaaS的支撑存在如下一些问题。

(1)对MT的支持不充分

租户信息在业务逻辑与数据处理之间的传递由SaaS承担或由PaaS协助承担,如SalesForce.com的CRM由相应的SaaS模块承担MT管理^[4],Google的MT管理则由PaaS平台的GAE提供^[5],然而对于运行在GAE上的第三方SaaS应用(the 3rd party software vendor,3PSV)并不完全支持(AWS和Windows Azure等平台存在同样的问题)。

(2)对可定制化性的支持不充分

将可定制性分为两级:按需求选择功能模块组建应用系统的宏观和按租户设定调整数据项数的微观,则基于元数据驱动的MDSA^[6]、基于模板和有效描述符定制化的方法^[7]主要是微观的;EasySaaS^[4]、CollabSoft^[8]主要是宏观的。基于能力模型的可定制化方法^[9]则兼顾宏观与微观。由上述研究发现,个性化定制是需要语义关联的。微观所涉及的语义关联度较深,通常由SaaS自身解决。宏观可定制性可通过引入领域模型与领域知识库完成,由PaaS承担此项任务将一劳永逸,而此项任务正是上述提及的当前主流平台所没有涉及的。

(3)对可扩展性的支持不充分

当前的IaaS利用虚拟化技术实现资源复用,屏蔽硬件差异,达到随需进行硬件扩充的目的,可满足SaaS对资源使用的较高要求。但它们之间有多层软件间隔,使IaaS对于SaaS的感知程度较低,造成SaaS很难依靠IaaS层实现自动扩展与负载均衡。而PaaS因提供SaaS的开发环境和运行环境^[3]可以感知它们,本应对SaaS的自动扩展提供充分的支持,但主流的PaaS缺少对领域应用的支撑,使得SaaS开发人员只能利用PaaS的API完成“自动”扩展,且以虚拟机为单位,提高了开发难度,造成了资源的浪费。事实上,引起过载的可能仅是某个模块、页面或构件(本文称之为热点)。另一个问题是,开发者与使用者都必须事先对资源的使用有所规划,这无疑提高了开发与使用的门槛,

与云计算理念的初衷相悖^[10]。

为此,本文提出了BIRISF(BIRIS framework),其以SaaS型构件(SaaS component,SC)为管理单位,提高了管理的精细度,易于捕获热点,能够及时进行扩展,减少资源浪费,同时建立领域模型库与知识库,用以支持SaaS在宏观和微观两个层次上的深度定制,并在BIRISF调度模型与散列一致性反馈负载均衡算法(consistent hashing feedback algorithm,CHF)的配合下,利用三阶段动态扩展与部署(three stages of dynamical extending & deploying,TSDED),合理地解决SaaS在功能与性能上的问题。最后,经过在国际海运物流领域的应用测试,证实了BIRISF在开发、运维及性能方面均优于经典的多层架构。

2 BIRIS 框架

2.1 整体框架

双边资源整合服务(bilateral resource integration service,BIRIS)是通过引入第三方服务平台来聚集大量顾客与服务提供者,既为顾客提供者整合顾客资源,又为顾客整合来自不同提供者的服务资源的服务模式^[12]。BIRISF是体现BIRIS思想的基于开源技术构建的服务框架,支持多租户服务系统的构建和运行。由于云计算的关键在于组件的属性,即可复用性、可替换性、可定制化性和可扩充性^[9],因此,BIRISF完全是由SC组成的,SC是管理的最小粒度,通过引入领域模型来增强SC之间语义的关联,实现了对SaaS的可定制化、动态部署、扩展、调度及MT等属性的支持。BIRISF的总体框架如图1所示,具体介绍如下:(1)~(2)领域专家或3PSV可通过平台构建或完善元模型库、模型库和领域知识库;(3)~(4)3PSV可对开发的SC进行注册并部署;(5)租户可以按照自己的需求、平台及SC提供商或领域专家的相关规则定制应用系统;(6)为其客户提供相关服务。

平台具备租户感知功能,在监控及管理组件的配合下,完成SC调度与负载均衡。模型解析引擎可以识别并解析模型,实现租户对于应用系统的定义。SC的部署分为两种:手动部署和自动部署,其中,后者需要在资源管理及扩展工具的配合下完成。因为BIRISF也是符合本框架构建与运行要求的规则,所以在初始启动时,领域知识库、构件库、模型库及元模型库并不为空。

2.2 BIRISF的基本元素

BIRISF管理和构成的基本单位都是SC,这不但提高

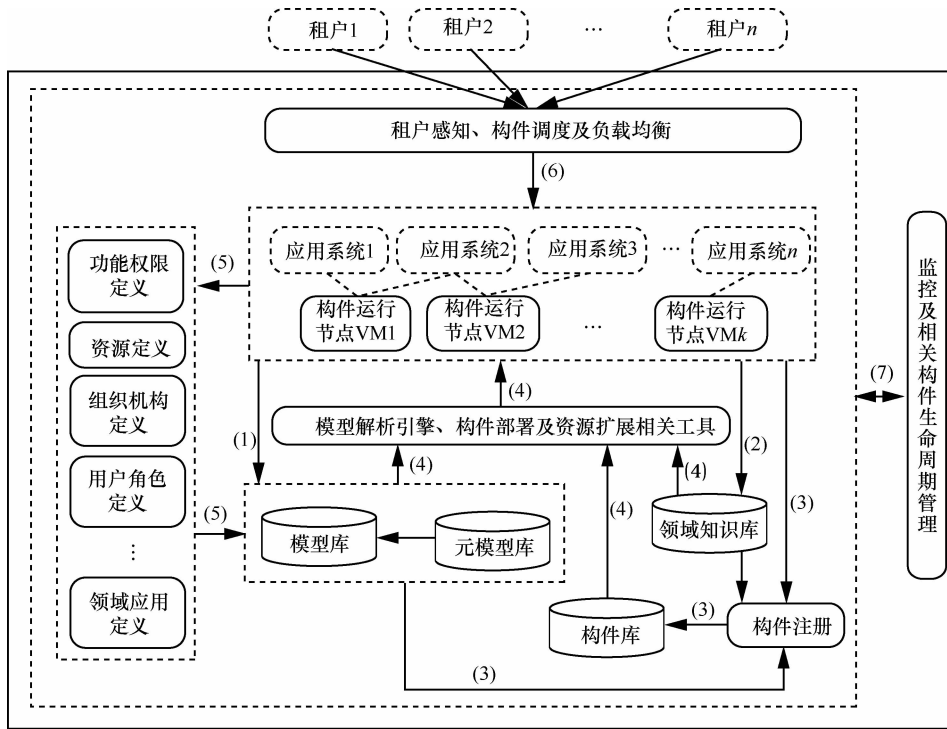


图1 基于模型驱动的BIRISF框架原型

了管理的精细度,还对MT、可扩展性、可定制性有了更好的支持,另外,SC可以封装成Web Services对外服务。为了便于说明,这里给出SC的一些形式化定义。

定义1 (SC模型) 一个SC代表在某个特定领域里相对独立的功能实体,表示为 $SC=(SCID, R, DS, Config, f_E, f_P, f_R)$,其中, $SCID$ 为构件的唯一标识; R 为对该构件的访问规则,含个性化定制的规则、数据访问协议等; DS 为该构件可以访问的所有数据的信息空间; $Config$ 为构件的个性化定制描述, $Config=(customerid, config)$ 且 $config \in R$; f_E 为访问规则的解析函数; f_P 为构件对外提供的服务集合; f_R 为构件运行所需要的服务集合。 $R, Config$ 与 f_E 降低了构件间的耦合度并满足SC在微观可定制性方面的需求,更适合MT。通用构件模型和工作流模型作为模型被存储在模型库里,作为识别构件、执行工作流的标准,组成构件的复合元素 $DS, Config$ 作为构件的元模型存入元模型库中。

约束条件介绍如下。

- $\forall config_i \in R, metadata \in DS$ 必有 $f_E(SCID, config_i, metadata) \neq \Phi$, 即对于给定的构件 $SCID$,任意给定一个符合规则 R 的配置,都能够找到一种个性化的配置方案。

- $\forall config_i \in R, metadata, data \in DS, f_i \in f_P$, 必有 $f_i(data, f_E(SCID, config_i, metadata)) \in DS$ 或者 $f_i(data, f_E(config_i, metadata, SCID)) \neq \Phi$, 即任意给定一个符合规则 R 的配置及合法的输入 $data$,所得到的结果要么在构件的信息空间中,要么为空。
- $\forall f_i \in f_P$, 必有 $f_i = f_E(SCID, config_i, f_R, DS)$ 且 $config_i \in R$, 这表明构件对外所提供的功能服务必须满足其运行条件 f_R 、符合定制规则 R 、在其数据空间 DS 上可由 f_E 解析而获得。

定义2 (构件的复用度) 构件 SC 的复用度 $SCR(f_x, \Omega)$ 定义为在所有构件所提供的服务空间 Ω 中,能够被复用的机会的大小,分为绝对复用度 $SCR_A(f_x, \Omega)$ 和相对复用度 $SCR_R(f_x, \Omega)$, f_x 代表 f_P 或 f_R 。 $SCR_A(SC, \Omega)$ 可以通过 SC 被其他构件引用的总次数来度量,表示为 $SCR_A(f_x, \Omega) = \sum_{j=1}^n f_x(SC)$, $f_x \in \Omega$ 。当 $f_x = f_P$ 时, $SCR_A(SC, \Omega)$ 越大,表明 SC 被复用的次数越多,所能够适应的业务需求的变化就越灵活;当 $f_x = f_R$ 时, $SCR_A(SC, \Omega)$ 越大,其所需要其他构件提供的服务越多,构件的组合程度越高,应用层次越高。 $SCR_R(SC, \Omega)$ 描述了 SC 可被复用的机会的大小, $T(\Omega)$ 表示服务总量, $SCR_R(SC_i, \Omega) = SCR_A(SC_i, \Omega) / T(\Omega)$ 。

定义 3 (原子 SC 模式) 当 $R=\emptyset$ 且 $f_R=\emptyset$ 时, SC 称为原子构件。因为 $Config \in R$, 所以 $Config=\emptyset$, 由于不需要解析任何规则, f_E 失去了存在的意义, 即 $SC_{\text{atom}}=(SCID, \emptyset, DS, \emptyset, \emptyset, f_E, \emptyset)$ 。

此定义表明原子构件是非组合且不可定制的, 只能对外提供相应服务。

定义 4 (框架 SC 模式) 构件 $SC=(SCID, R, DS, Config, f_E, f_P, f_R)$ 称为框架构件 (SC_F), 当且仅当 $\forall f \in f_P$ 时, $SCR_R(f, SC_F)=1$, $\forall f \in f_R$ 时, $SCR_R(f, SC_F) < 1$ 。

框架 SC 满足“共同复用”原则^[13], 将具有相同需求的构件聚合在一起, 这与传统的面向对象“框架”的概念类似。非框架构件, 称为业务构件 (SC_B)。为开发和实现的方便, 将构件按照 OMG 3 层结构分为 UI (user interface)、BL (business logical computing)、OR (object relationship map) 3 种类型。

2.3 应用服务系统的构成

BIRISF 允许 3PSV 按照构件模型开发出符合规范要求的 SC, 并进行注册。因此, BIRISF 可以聚集大量面向领域的 SC。租户作为具有共性需求的外包用户群体, 可根据自己的需求选择并租用 SC, 在 BIRISF 上制定出己的应用系统, 称为 iSC (internet service center)。每个 iSC 必须建立一个或多个功能项, 并与选定的 SC 相关联, 方可被使用。同一构件可与多个租户的功能项关联, 即实现对 MT 的支持。租户是 iSC 的拥有者, 在 BIRISF 中注册为组织, 并可建立一套属于自己的组织管理体系 (如角色、等级等), 在 R 和 $Config$ 的约束下, 通过功能使用系统。BIRISF 的组织方式是与模型相关的。iSC 可以注册两类模型: 领域模型和资源模型, 领域模型建立起 SC 之间的关联, 有效组织系统运行; 资源模型把现实资源映射成系统可以管理的虚拟资源, 供消费者订阅并使用。BIRISF 赋予 iSC 充分的权力, 包括可以制定自己的主页、拥有自己的用户管理等。由此可知, BIRISF 在宏观与微观 $f_E (config, metadata, SCID)$ 的可定制化与 MT 上, 对 SaaS 给予的支持是充分的。

3 关键技术

BIRISF 针对 SaaS 应用在定制、部署、调度和管理等方面所带来的新挑战, 提出了一种通用的解决方案, 有效避免了每个运行在 BIRISF 下的 SaaS 应用对这些问题的重复考虑。下面分别就调度模型、负载均衡算法及动态扩展与部署等关键技术对 BIRISF 做进一步分析与探讨。

3.1 调度模型

由于负载压力及性能方面的要求, SC 的副本可能部署在满足其资源要求的多个节点上, iSC 中功能/服务项与相关的 SC 可能会存在一对多的映射关系, 在服务请求到达时, 需要 BIRISF 能够及时找到合适的 SC 响应服务请求。为此, 引入调度引擎 schedule 配合 f_E 共同引导 SC, 其中, $schedule, f_E \in \{SC_F\}$ 。如图 2 所示, iSC 建成后, 用户由 BIRISF-MFrame 登录 iSC, 通过 iSC 中的功能项向 schedule 发出请求, 调用 SC。

schedule 分为两部分: BL 和 OR。BL 负责汇总来自 OR 的领域知识、收集节点负载情况并做出决策, 将服务请求转发给选中的 SC 实例; OR 负责查询知识库和运行态 SC 库, 将符合要求的 SC 集合反馈至 BL。在决策的过程中, schedule 能发现异常的 SC, 重新转发服务请求, 因而具有一定的容错性。此处的运行态 SC 库不同于构件库, 是指运行于各节点上 SC 实例的集合。在数据处理方面, BIRISF 采用“三级两层”数据处理模式。所谓“三级两层”, 即 OR → 数据库映射服务组件 (memory database-global database map SC, MGDBM-SC) → GOR (global database-object relationship map) 3 级、内存数据库 (memory database, MMDb) → 全局数据库 (global database, Global-DB) 两层。“三级”使 MT 数据处理对于 SC 的开发者具有一定的透明性, 这项工作由相应的 SCF 与 MGDBM-SC 处理, 开发时只需调用 MGDBM-SC 所提供的接口即可; 两层数据库中, 内存数据库用于多租户数据隔离与缓冲, 提高效率, 全局数据库用于存储多租户数据的副本, 保障数据的安全。数据一致性与可靠性由 GOR 负责。schedule 的引入使 BIRISF 具有一定的开放性, 保障了 BIRISF 在宏观和微观两个层次上的可定制性, 支持 BIRISF 的动态可扩展性。

3.2 散列一致性动态反馈算法

根据第 3.1 节中的调度模型, 采用集中式策略将每个节点的计算负载和通信发送到一个指定的节点, 该节点负责收集、处理全局的负载信息, 并做出全局的负载均衡决策。本文参考散列一致性算法 Consistent Hashing^[11]与随机性反馈调度算法 Pick-KX^[13], 提出了改进的散列一致性动态反馈算法处理 BIRISF 中的负载平衡问题。

3.2.1 负载能力的计算

在云计算中可利用虚拟化技术屏蔽硬件资源之间的差异, 因而可以统一计算能力的标准。计算能力的度量以节点 (虚拟机) 为单位, 主要考虑 CPU、内存、I/O、进程/线程数

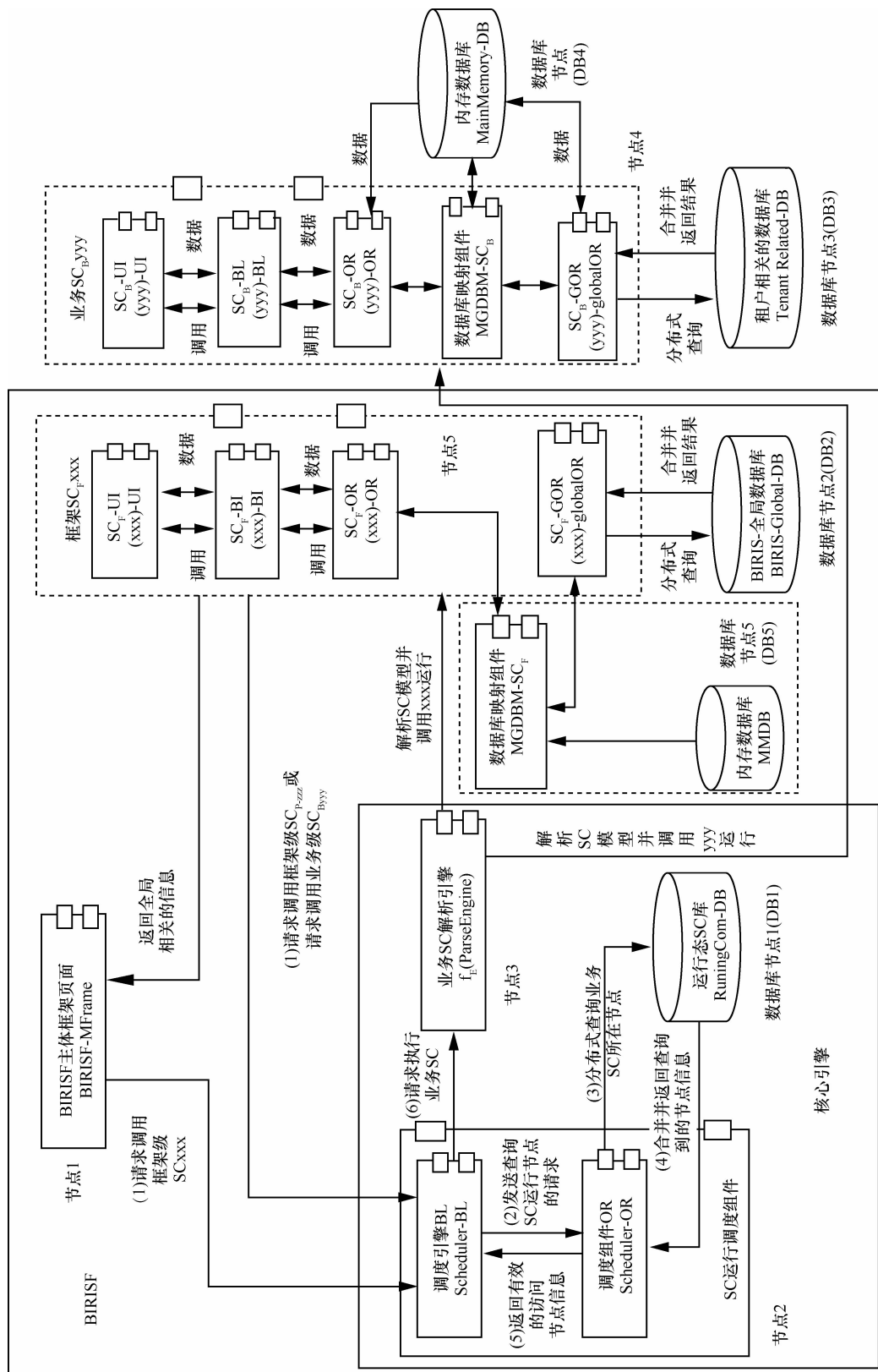


图2 SC 调度模型

和带宽 5 项参数,本算法允许使用者依据实际情况进行增减。另外,增加一个条件假设:按上述 BIRISF 的设计要求,所有 SC 都是受控的,因此,可以尽量将相互依赖的 SC 部署在相同的节点上,则任务之间的路径可以忽略。设在 BIRISF 中有 n 台虚拟机 $S=\{S_1, S_2, \dots, S_n\}$, 其中 S_i 的负载可描述为:

$$load(S_i) = (w_1, w_2, \dots, w_k) \begin{bmatrix} load(cpu_i) \\ load(mem_i) \\ load(io_i) \\ load(bw_i) \\ load(td_i) \\ \vdots \\ load(x_i) \end{bmatrix} \quad (1)$$

其中, $\sum_{j=1}^k w_j = 1, i=1, 2, \dots, n$ 。 $load(cpu_i)$ 、 $load(mem_i)$ 、 $load(io_i)$ 、 $load(bw_i)$ 、 $load(td_i)$ 、 $load(x_i)$ 分别表示节点 i 当前的 CPU、线程/进程占用率、内存、磁盘 I/O、带宽及其他影响因素, w_j 代表每个负载项影响因子,由实验测得。支撑 SC 运行的 Java 虚拟机或 Web 容器会有一个最大并发线程数 TD_{max} ,不同容器的 TD_{max} 可能不同,因此用线程/进程占用率衡量当前容器的负载情况较为合理,即 $load(td) = TD_{current}/TD_{max}$, $TD_{current}$ 表示当前并发的线程或进程数。容器已经饱和而资源占用率不高的情况在云环境下时有发生,因此在计算负载时,要加入线程/进程占用率。是否能够给某个节点分派任务,取决于该节点的剩余负载能力,计算式如下:

$$weight(S_i) = \frac{1 - load(S_i)}{\sum_{i=1}^n (1 - load(S_i))} \quad (2)$$

由式(2)可知, $weight(S_i) \in (0 \sim 1)$, 值越大,说明该节点的剩余负载能力越强。当系统实施时,还会设一个负载阈值 λ , 当 $weight(S_i) < \lambda$ 时,节点超载,不会再给该节点分派任务。

3.2.2 散列映射关系

构造一个 $0 \sim key_{max}$ 散列空间 τ , 按节点权重划分 τ , 则每个位置的计算方法如下:

$$\begin{cases} point(i) = 0, & i = 0 \\ point(i) = point(i-1) + [key_{max} \times weight(S_i)], & i \neq 0 \end{cases} \quad (3)$$

其中, $i=0, 1, 2, \dots, n$ 。用 $P(S_i)$ 表示节点区间, $P(S_i) = (point(i-1), point(i)]$, 则按式(4)设置散列函数的值。

$$hash(j) = S_i, j \in P(S_i) \quad (4)$$

由式(3)、式(4)可知每个节点在 τ 上的分布情况,如图3所示。由此可知,节点的剩余负载能力越大,即 $weight(S_i)$ 值越大, τ 中的范围也就越大。

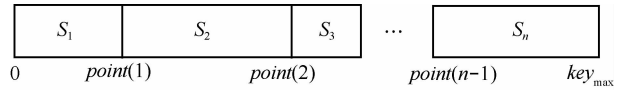


图3 节点在散列空间的分布

对每一个请求任务都赋予一个随机生成的序列号 seq , 并且使 $seq \in \tau$ 。将 seq 代入式(4), 即可得到 S_i , 从而将这个任务分派到节点 S_i 执行。根据 seq 的随机性, 它可能落到 τ 任意一个位置, 而范围越宽的节点, 获取这个任务的概率也越大。显然, 这一算法策略有利于整个系统的负载平衡。节点增减时, 因为权值总重不会超过 1, 因此不会影响到散列空间的大小, 节点的增加会使节点之间的距离变短, 任务落到各个节点的机率均会增加; 节点减少会使节点之间的距离变长, 依据 seq 的随机性, 任务分布仍然可以保持均匀。与其他算法比较, 它还有运算简单、调度效率高等特点, 因而更加适合于吞吐量高、实时性要求高、资源波动频繁的云计算环境。

3.2.3 负载均衡度量

本文采用均方差法对当前负载情况进行评估, 并适时做出调整。由每台服务器的综合负载可得到平均负载 $avg\ l$, 云计算环境下的负载均衡度量值 B 定义如下:

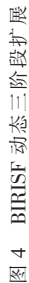
$$avg\ l = \frac{\sum_{i=1}^n load(S_i)}{n} \quad (5)$$

$$B = \frac{\sqrt{\sum_{i=1}^n (load(S_i) - avg\ l)^2}}{n} \quad (6)$$

其中, n 是主机数量, B 是负载均衡程度, B 越大, 负载均衡程度越小。当 B 大于阈值 B_0 时, 将负载最大的节点上的占用资源最多的任务迁移到负载最小的节点上。

3.3 三阶段动态扩展策略

动态扩展包含两方面的含义: 功能扩展与性能扩展。前者允许租户从小规模应用开始, 按需求自由订退 SC; 后者随着租户用户数量的增加逐步扩大, 需要更多资源投入支撑 SC 的运行。根据租户的商业发展模式, 把 SaaS 系统的扩展性问题分为 3 个阶段: 基于功能的扩展、基于性能的扩展及多版本共存, 如图 4 所示。



(1)基于功能的扩展

构件提供者在注册 SC 时,一方面要把 SC 实体文件运用传送引擎通过文件传送通道上传至虚拟节点,另一方面在信息数据库登记构件的元数据信息,如图 4 中(1)、(2)所示。租户通过 SC 构件视图,浏览并选择 SC 建立功能项,将功能项与相关构件关联,完成订阅过程。如果 BIRISF 通过业务构件监控组件感知该 SC 是首次被订阅,则选择适当节点部署该 SC,并利用数据库关联引擎建立租户数据库,否则只进行数据库的建立。随后,租户便可通过功能项使用该 SC 所提供的服务。这是一个按需订退的过程,当商业模式基本成熟后,其变化就不会很大。

(2)基于性能的扩展

商业模式稳定后,会有部分 SC 被大量访问而形成热点,BIRISF 以 SC 为单位进行调整。当 SC 监视组件发现某一 SC 访问量过载(超过某一阈值)时,会选择合适的节点将副本部署上去,然后再按负载均衡的方法将负载分流。当所有节点的负载均达到上限时,BIRISF 会创建新的节点进行部署。

(3)多版本共存

多版本共存实际上是上述两种扩展的一个综合应用。当租户发现有新版本 SC 出现时,为得到更好的性能与更完善的服务,退订原来的 SC,订阅新的 SC。但如果老版本仍有租户使用,则 BIRISF 可以保证新旧版本同时共享。

3.4 动态部署

在 SC 部署时,需要知道 SC 之间的依赖关系,避免因缺少必要的支持而不能运行或出错。在 SC 模型定义中, f_R 、 f_P 及 $Config$ 都表达了构件之间的关系,因而在 SC 注册时,SC 的依赖信息必须记录到构件库中,从而得到 SC 关联矩阵,见表 1。同样,由运行态 SC 库可得到当前部署矩阵,见表 2。

表 1 SC 关联矩阵

节点 $f_R(SC_i)$	SC_1	SC_2	SC_3	SC_4	...	SC_m
SC_1	1	0	1	1	...	0
SC_2	0	1	1	...	0	
SC_3	1	0	1	0	...	0
SC_4	1					
...			...			
SC_m	0	1	0	1	...	1

表 2 SC 当前部署矩阵

节点构件	N_1	N_2	N_3	N_4	...	N_n
SC_1	1	0	1	1	...	0
SC_2	0	1	1	0	...	1
SC_3	1	0	1	0	...	0
SC_4	1	0	0	1	...	0
...			...			
SC_m	0	1	0	1	...	1

BIRISF 对 SC 有完全的控制权,可在资源条件允许的情况下,将其所依赖的 SC 均部署在同一节点上,以提高服务质量。此即使散列一致性反馈负载均衡算法中的假设条件成立。在这种条件下,也要考虑如下 3 个因素。

(1)Web 容器所能容纳的最大进/线程。尽管这与承载 SC 的数目没有直接关联,但它具有突破进/线程上限的潜在可能。

(2)限定某个 SC 依赖的上限。对于某个 SC 的依赖越大,它成为瓶颈的可能性就越大。

(3)满足条件(1)和(2)的情况下,求 SC 部署的最小代价,即移动的 SC 数目最小,在拥有该 SC 所依赖的 SC 数目最多节点上部署该 SC。可按式(7)选取节点:

$Node = \min$

$$\left\{ Sum_i(f_R(SC_i)) - \sum_{j=1}^{m_k} Sum_j(\delta_{f_P(SC_i)=f_R(SC_j)}(f_P(SC_j))) ; k=1,2,\cdots,n \right\}$$

其中, $Sum_i(f_R(SC_i))$ 为 SC_i 需要其他 SC 提供服务的总数; m_k 为第 k 个节点已部署的 SC 数目。 $(\delta_{f_P(SC_i)=f_R(SC_j)}(f_P(SC_j)))$ 表示从 SC_j 对外提供的服务中找出为 SC_i 所提供的服务的数量,当 k 从 1 取到 n 时,则可以得到每个节点还需为 SC_i 提供的服务数目,它表达了除 SC_i 外还需部署的 SC 数量,找到最小的一个 $Node$,把 SC_i 及其剩余依赖服务部署到该节点上。

4 实例研究与结果分析

4.1 测试环境

为了便于比较,用两套基本等值硬件作为基础设施,一组安装经典多层非云系统平台 CERP 3.0;另一组开发工具与运行环境基本与前者相同,为加强构件独立运行、即插即用等性能,引入 OSGi 技术,并在 Web 应用服务器中部署 OSGi 容器,见表 3。



表 3 经典多层架构与云架构软硬件方案对比

方案编号	平台架构	开发工具及运行环境	硬件资源及分配方案	使用模式及使用对象
1	CERP 3.0	Extjs+Spring+dwr+Hibernate+非集群数据库+Web 应用服务器	(Intel Xeon Processors 4 核 @ 3.2 GHz + 内存 8 GB + 硬盘 500 GB + 网卡 1 000 Mbit/s)×3; 其中,应用服务器 2 台;数据库服务器 1 台	单租户;场站:场站业务外+部分货代业务
2	BIRISF 1.0	Extjs+Spring+dwr+Hibernate+OSGi+集群数据库+Web 应用服务器+CloukStack 4.0	((Pentum Dual-Core CPU E5700 @ 3.00 GHz 或 AMD Athlon 64×2 Dual Core 5200+ @2.7) + 内存 4 GB + 硬盘 320 GB + 网卡 100 Mbit/s) ×14; 其中,数据库服务器 2 台;管理节点 2 台;主机 6 台;主存储器 1 台;备用存储器 1 台;用作扩展的后备机 2 台;其中主机与主存储器做成一个 pod	多租户;场站,货代,船公司:场站业务+船公司业务+货代业务

4.2 实例方案验证

4.2.1 实例方案

由于 ESA (enterprise software & application) 软件基本上都以单据、报表及其处理为中心,因此,以单据为单位进行模块或构件划分,统一对比标准。方案 1 为传统的单租户模式,服务于某家大型场站。其场站业务约有 81 个模块,15 个流程模型;货代业务约有 31 个模块,9 个流程模型,为了缓解因过载而产生的服务质量下降,采用反向代理+双机并行模式,即两台应用服务器所安装的软件一样,由反向代理确保负载平衡。方案 2 为 BIRISF 模式 (BIRISF-P),涵盖国际海运物流信息管理三大主要系统:船公司 (50 个单据模块,16 个流程)、场站和货代,经过 SaaS 模式的改造,提取出公用部件,减少了构件总量。与传统开发模式相比,构件数目从 162 个降低到 107 个,缩减了 34%,有效地降低了开发成本。

由于构件数量众多,本文篇幅有限,只将 BIRISF 平台及其运行实例的主体部分展示出来,本示例共分 4 个部分:SaaS 构件模型及实体、运行支撑平台、应用系统及相应的组织模型。符合 BIRISF 规范的构件分为系统 (框架) 构件和业务构件两部分,系统构件又分为引擎构件和管理构件两部分,它们部署后共同组成运行支撑平台。业务构件可以按领域分为多种类型,本文中着重以国际物流为例,部署后形成多个 iSC 应用系统,此处设置场站、货代和船公司各两个。前文所述的领域知识及相关模型具体表现形式更多体现在组织模式上,它们部署于平台及应用系统中,把那些彼此相对独立的 SC,以松散耦合的方式有机地结合起来。按照 BIRISF 的运行要求,其虚拟机创建清单见表 4。

表 4 虚拟机创建清单

作用	配置	数量 (个)
运行 BIRISF 主体框架	CPU 2 GHz, 内存 2 GB, 硬盘 40 GB	2
运行构件库管理	CPU 2 GHz, 内存 1 GB, 硬盘 100 GB	2
运行国际海运物流业务 SC	CPU 2 GHz, 内存 2 GB, 硬盘 40 GB	6

4.2.2 负载项影响因子的设定

对于 ESA 软件来说,每项任务的数据传送量并不大,而计算量大,选定 100 个偏重计算的任务反复实验。初始化时,影响因子 $\{w_1, w_2, w_3, w_4, w_5\}$ 分别设定为 $\{0.5, 0.2, 0.1, 0.1, 0.1\}$, 从 w_1 开始每次递减 0.05, 其后影响因子依次增加, 即 $\{0.45, 0.25, 0.1, 0.1, 0.1\}$ 、 $\{0.45, 0.20, 0.15, 0.1, 0.1\}$ 等, 测试结果如图 5 所示。从图 5 中不难观察到, 磁盘 I/O 与带宽权值变化时, 对响应时间的影响不大, 如第 4、5 节点为 $\{w_4, w_5\} = \{0.15, 0.1\}$ 、 $\{0.1, 0.15\}$, 第 8、9 节点为 $\{0.15, 0.15\}$ 、 $\{0.1, 0.2\}$ 。当达到最低点时, 再从最低点 (次数为 19, 影响因子为 $\{0.3, 0.2, 0.2, 0.1, 0.2\}$, 响应时间为 2 111 ms) 回升, 即下一组影响因子为 $\{0.3, 0.2, 0.25, 0.1, 0.15\}$, 以此类推, 经过数百次实验, 规律基本相同, 因此可设定系统运行的初始影响因子为 $\{0.3, 0.2, 0.2, 0.1, 0.2\}$ 。此值并非一直不变, 监控组件会定期采样, 根据具体情况进行调整。

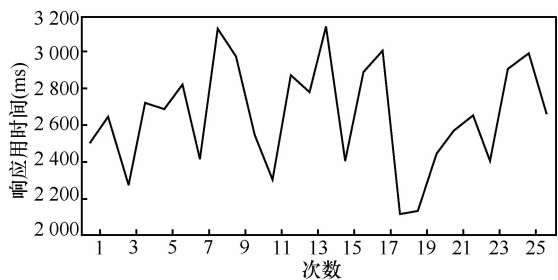


图 5 负载项影响因子测试结果

4.2.3 实例结果

图 6 展示了并发数量对负载均衡 (计算 B 值, 值越小, 负载越均衡)、响应时间的影响。基本规律是随着负载的增加, 响应时间越来越长。负载均衡越好的系统, 平均响应时

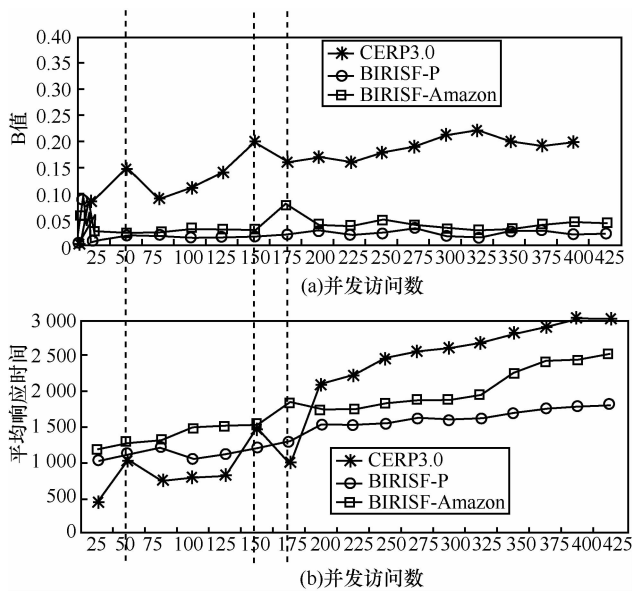


图6 并发访问数量与负载均衡、平均响应时间测试结果

间增长得越慢,同时能承受的负载压力也越大。在负载均衡和平均响应时间上,因为扩散了热点、使用了 CHF 算法和 TSDED 策略,BIRISF-P 表现最好,BIRISF-Amazon 稍逊于它,主要是网络原因,两者运行的都是 BIRISF 1.0,应该是无差别的。CERP 3.0 在 B 值上表现较差,是因为表面上公平的轮循分配方式隐藏了内部运算量分配不公的事实。图 6 中虚线标识的峰值就是因为轮循分配中不断有运算量较大的任务被分派到相同的节点,它也促成平均响应时间的陡然提高。CERP 3.0 在平均响应时间上的增长速度也非常快,在达到 425 的并发时,甚至会有拒绝服务的现象发生,然而经检测,此时硬件资源综合利用率不超过 40%。这说明了 CERP 3.0 在两方面存在问题:一方面,没有考虑软件应用服务器与硬件资源的配置,应用服务器进程数已满,而资源却未充分利用,并不能进行动态扩展;另一方面,没有考虑热点问题,可能热点都集中在一台主机上,而另一台却在大部分时间内闲置,调度算法过于简单。

从图 6 中可以看出,BIRISF 借助其调度模型、散列一致性反馈算法及三阶段动态扩展与部署等策略,成功地解决了上述问题。在目前配置下,BIRISF-P 开启 6 台业务虚拟机,400 以上并发,服务质量并无明显下降,峰值时,资源利用率可达 70%左右。

5 结束语

本文提出了基于双边资源整合的框架 BIRISF,可以聚集第三方提供的 SC 资源。BIRISF 中动态调度的模型、CHF

算法和 TSDED 策略,经检验其性能满足 SaaS 应用系统的多方面需求,解决了当前云平台对于第三方所开发 SaaS 应用系统相关的支撑与管理问题,取得了较好效果。下一步将在 BIRISF 上构建多领域多应用的 SaaS 系统,用以研究这些应用的共性问题,进一步完善 BIRISF。

参考文献

- 1 Mell P, Grance T. The NIST Definition of Cloud Computing. National Institute of Standards and Technology, 2011
- 2 Tsai W T, Huang Y, Bai X Y, *et al.* Scalable architectures for SaaS. Proceedings on 15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), Shenzhen, China, 2012:112~117
- 3 Lenk A, Kelms M, Nimis J, *et al.* What's inside the cloud? An architectural map of the cloud landscape. <http://www2.computer.org/portal/web/csdli/doi/10.1109/CLOUD, 2009>
- 4 Weissman C D, Bobrowski S. The design of the force.com multitenant internet application development platform. Proceedings of the SIGMOD, Providence, Rhode Island, 2009: 889~896
- 5 Google App engine. <http://code.google.com/appengine/2011, 2011>
- 6 林海略, 韩燕. 多租户应用的性能管理关键问题研究. 计算机学报, 2010, 33(10):1881~1895
- 7 Mietzner R. Using variability descriptors to describe customizable SaaS application templates. Technical Report Computer Science, 2008, 1(2):1241~1247
- 8 Doelitzscher F, Sulistio A, Reich C, *et al.* Private cloud for collaboration and e-Learning services: from IaaS to SaaS. Computing, 2011, 91(1): 23~42
- 9 Sun W, Zhang X, Guo C J, *et al.* Software as a service: configuration and customization perspectives. Congress on Services Part II, Services-2, IEEE, 2008(1): 18~25
- 10 Fox A, Griffith R, Joseph A, *et al.* Above the clouds: a berkeley view of cloud computing. Department Electrical Eng and Computing Sciences, University of California, Berkeley, Rep UCB/EECS, 2009(28)
- 11 Karger D, Lehman E, Leighton T, *et al.* Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide Web. Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, ACM, El Paso, Texas, 1997: 654~663
- 12 Wang Z J, Xu X F. Bilateral resource integration service mode for value innovation. Computer Integrated Manufacturing Systems, 2009, 15(11):2216~2225



表 2 实验数据

时间点 i	扫描威胁	总体威胁	网络安全态势值	网络安全趋势(ORAR 算法)	网络安全趋势(改进 ORAR 算法)
335	轻微(690 次)	一般(19 721 次)	安全程度低(2 542)	形势好转	形势好转
336	轻微(573 次)	一般(18 754 次)	安全程度中(2 231)		
421	轻微(628 次)	一般(18 401 次)	安全程度低(2 371)	形势恶化	形势好转
422	轻微(644 次)	一般(17 148 次)	安全程度低(2 393)		

的不同,使算法在预测点 B 时出现错误,而在改进算法下,因为充分考虑到网络趋势变化情况,避免了预测规则的孤立,从而得出了较为精确的结果。但是,在预测规则与趋势变化结果相冲突时,需要引入人为影响,这在一定程度上降低了算法的客观性。

4 结束语

将网络安全态势感知和云理论结合起来,针对现有的基于云的网络安全态势预测规则挖掘算法予以改进,使算法与实际结合更加紧密,结果更加精确。因为实验中验证的数据较少,因此对网络安全态势的发展预测还仅限于局部。当提供的数据量增加时,预测规则的数量将同时得到提高,预测结果也将更加精确。但是,如何将算法应用于实际以及如何解决算法中两方结论冲突时的决策问题,仍然需要做进一步的研究。

参考文献

- 1 Chang K C, Saha R K, Yin X Y. A linear predictive bandwidth conservation algorithm for situation awareness. http://volgenau.gmu.edu/~kchang/publications/conference_pdf/CON_9_A_LINEAR.pdf,1998
- 2 任伟. 网络安全态势评估智能化研究. 上海交通大学硕士学位论文,2007

- 3 宣蕾. 网络安全定量风险评估及预测技术研究. 国防科学技术大学博士学位论文,2007
- 4 李德毅,杜鹤. 不确定性人工智能. 北京:国防工业出版社,2005
- 5 郝树勇. 基于云的网络安全态势预测研究. 国防科学技术大学硕士学位论文,2010
- 6 叶琼,李绍稳,张友华等. 云模型及应用综述. 计算机工程与设计,2011,32(12):4198~4201
- 7 周文秀. 关联规则挖掘算法的研究与改进. 武汉理工大学硕士学位论文,2008

[作者简介]



朱殊,男,武警工程大学信息工程系硕士研究生,主要研究方向为网络安全。

刘丽霞,女,博士,武警工程大学信息工程系教授,主要研究方向为算法优化和嵌入式系统应用。

邱晓华,男,武警工程大学信息工程系助教,主要研究方向为计算机网络。

(收稿日期:2013-07-20)

(上接第 81 页)

- 13 Martin R C. Agile Software Development: Principles, and Practices. Prentice Hall PTR, 2003

[作者简介]



张小东,男,哈尔滨工业大学博士研究生,主要研究方向为云服务平台框架及其关键技术。

战德臣,男,哈尔滨工业大学教授、博士生导师,主要研究方向为服务工程与服务计算、软件工程。

聂兰顺,男,博士,哈尔滨工业大学副教授,主要研究方向为信息物理系统、资源调度、优化方法等。

孟凡超,男,博士,哈尔滨工业大学(威海)副教授,CCF 高级会员,主要研究方向为软件体系结构、云计算。

(收稿日期:2013-07-31)