一种提供容错性保障的药物设计网格环境

王永剑[1]，任一楠[1]，陈婷[1]，黄远强[1]，于坤千[2]，栾钟治[1]，蒋华良[2]，钱德沛[1,3]
(1.北京航空航天大学计算机学院，100191，北京；2.中国科学院上海药物所，201203，上海；
3.西安交通大学电子与信息工程学院，710049，西安)

摘要：针对药物虚拟筛选操作运行时间长、易出错等问题，设计并实现了药物设计网格(DDGrid)，有效聚合了中国国家网格环境中闲散的资源，形成了支持高通量药物虚拟筛选的网格环境.所设计的DDGrid网格实现了基于复杂事件处理技术的有状态故障检测机制，通过持续分析节点间交互的事件流，定位可能的故障场景并触发相应的补偿操作来减少故障导致的损失，为系统提供了容错性保障.实验结果表明，有故障检测机制可以通过监测筛选任务运行情况对超时阀值进行动态调整，使得任务实际完成时间和理论完成时间之间的差值保持在10% 以内，同时提高了底层计算资源的利用率.

关键词：虚拟筛选；容错；有状态故障检测；复杂事件处理
中图分类号：TP391.9 **文献标志码**：A **文章编号**：0253-987X(2009)12-0021-05

A Drug Discovery Grid Environment with Fault-Tolerance Support

WANG Yongjian[1], REN Yinan[1], CHEN Ting[1], HUANG Yuanqiang[1], YU Kunqian[2]
LUAN Zhongzhi[1], JIANG Hualiang[2], QIAN Depei[1,3]
(1. School of Computer Science, Beihang University, Beijing 100191, China; 2. Shanghai Institute of Materia Medica, CAS, Shanghai 201203, China; 3. School of Electronics and Information Engineering, Xi′an Jiaotong University, Xi′an 710049, China)

Abstract: In order to resolve the long running and error-prone problems in virtual screening, a drug discovery grid (DDGrid) is designed and implemented to establish a grid environment. The environment utilizes idle resources scattered over the CNGrid environment to support high-throughput virtual screening. The DDGrid implements stateful fault detection mechanism based on a complex event processing technology, and provides fault-tolerance support through continuously analyzing the exchanging event streams among different nodes, locating possible faulty situations and triggering the corresponding compensating operations to minimize the possible loss. Experimental results show that the fault detection mechanism keeps the difference between the actual task process time and the theoretical value within 10% through adjusting timeout threshold in the runtime, and improves the usage of underlying computing resources.

Keywords: virtual screening; fault-tolerance; stateful fault detection; complex event processing

药物设计早期阶段的焦点内容之一是如何从百万规模的分子数据库中找到有药理活性的化合物.虚拟筛选是近年来日益受到重视的一种药物设计手段，它基于分子对接技术寻找潜在的化合物，能够几个数量级地减少需要进行生物实验的候选化合物数量.然而，由于虚拟筛选对计算资源的需求远远超过了单个研究所或公司的承受范围，因此网格技术被广泛用来聚合分散的计算资源，为虚拟筛选操作提供计算基础设施.

药物虚拟筛选操作具有时间长、易出错等特点，

收稿日期：2009-03-17. 作者简介：王永剑(1981—)，男，博士生；钱德沛(联系人)，男，教授，博士生导师. 基金项目：欧盟FP6资助项目(045609)；国家高技术研究发展计划资助项目(2006AA01A106，2006AA01A124，2006AA01A118).

在系统设计和构建过程中,需要重点考虑如何支持筛选操作的长时间稳定运行,尽可能快速准确地定位故障,并执行必要的补偿操作来减少可能的损失.常见的网格中间件侧重于支持资源聚合,无法有效满足药物虚拟筛选对稳定性及可靠性的需求.

药物设计网格(DDGrid)是中国国家网格[1]支持的应用项目,目的是基于中国国家网格基础设施构建一个虚拟的计算网格环境[2],聚合互联网环境下闲散的计算能力,同时支持各种类型的分子对接工具,如 GAsDock[3]软件等,为用户提供在线的高通量药物虚拟筛选服务.

本文提出了 DDGrid 基于复杂事件处理技术的有状态故障检测机制,通过为 DDGrid 提供有效的容错性保障来支持虚拟筛选操作的长时间稳定运行,同时提高了系统的可靠性及底层资源的利用率.

1 药物设计网格系统结构

图 1 是药物设计网格系统结构,遵循 Master/Worker 结构[4],分为 4 个功能层:工作流引擎、控制节点、传输模块以及工作节点.为了提高系统的可靠性,我们添加了额外的有状态故障检测机制.

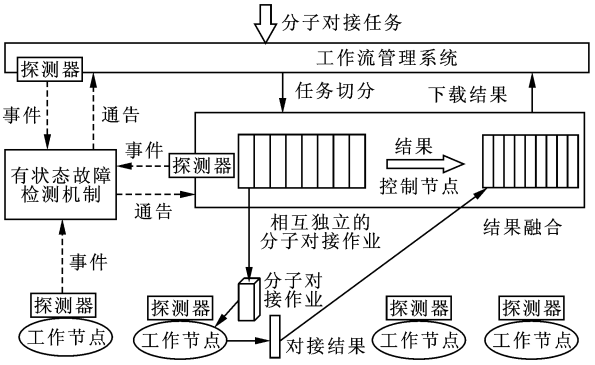


图 1 药物设计网格系统结构

2 容错性支持

常见的故障检测机制根据其手段可分为无状态和有状态 2 种:无状态故障检测通常基于单个事件特征来来达到识别故障的目的;有状态故障检测[5]是一种更有效的方式,通过聚合多个事件实例来构建故障上下文场景,进而识别故障场景.

本文提出一种基于复杂事件处理技术[6-8]的有状态故障检测机制(Cesar-FD). Cesar-FD 支持类 SQL 的数据模型和语法,加入对时间/长度滑动窗口扩展的支持,可以从连续的事件流中识别非法的事件序列.

2.1 事件模型

事件模型包括事件类型和事件实例 2 部分,进入事件处理系统的事件都应符合预定义的事件模型.

事件类型用来刻画系统的状态变化,由一系列属性字段组成,如编号、名字等,采用大写字母 A、B 等表示.事件实例用来描述某一个时间点的系统状态变化,是某一事件类型的实例对象,满足即时性、原子性等特征,采用 a_1、a_2、b_1、b_2 等表示,其中 a_1、a_2 表示事件类型 A 的不同实例对象.除此之外,每个事件实例还会被分配一个离散时域的时间戳.这里我们假设这些时间戳是在事件实例进入事件处理系统之前通过独立机制生成,能够反映事件实例之间真实的先后顺序.这个假设并不适用于所有场景,但是在 DDGrid 中是可以接受的.文献[9]详细讨论了事件实例之间的时序关系.

2.2 复杂事件处理引擎

复杂事件处理技术基于事件流和滑动窗口,支持对单源或多源实时事件及历史事件的有效处理.

DDGrid 使用自实现的复杂事件处理引擎 Cesar,其 Cesar 事件流引擎处理流程如图 2 所示. Cesar 基于注册的过滤条件对输入事件流$<a_1,b_1,c_1,c_2,b_2,c_3,\cdots>$进行分析,提取符合条件的事件实例的属性字段值构建输出事件$<d,e>$,其中事件 d 由事件实例 a_2 的 id 属性值 i_{d} 组成,事件 e 由事件实例 c_3 的 name 属性值 n_{ame} 组成.

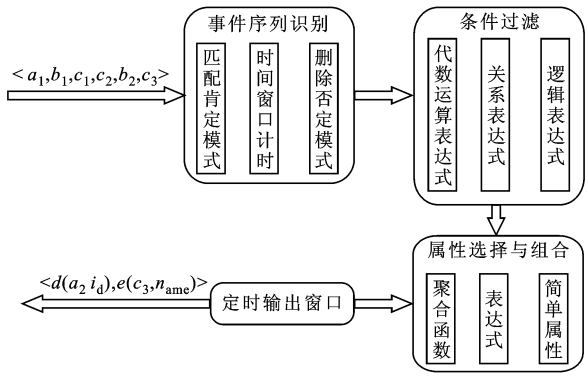


图 2 Cesar 事件流引擎处理流程

Cesar 将整个处理过程切分为若干相互独立的阶段来支持模块化的扩展.

(1)事件序列识别:负责检测符合预设事件序列且处于滑动窗口中的事件组合.实际上,事件序列识别仍然是采用基于优化的 NFA 识别技术实现的,类似于栈分配中识别 XML 模式的 PathStack 技术[10].

(2)条件过滤:负责将虽通过事件序列识别阶段但所包含事件的属性值并不符合要求的事件组合过滤掉.这一阶段可进行的过滤运算包含简单表达式、不等式比较、比较结果的逻辑组合等.

(3)属性组合与选择:负责将用户关心的事件属性进行提取和重组.

(4)输出:负责将属性组合与选择阶段的结果按照指定的频率进行输出.

上述各操作基于管道化方式执行:如果到达的事件与先前的事件组合在一起,符合指定的事件序列,这个事件序列会立即由事件序列识别阶段处理,再通过管道完成一系列后续处理,最终加入输出事件流中.

Cesar 支持类 SQL 语法,并提供滑动的时间/长度窗口扩展支持.Cesar 语言的语法结构如下

select <select_list>
usepattern <event_pattern_expression>
[where <search_conditions>]
[with <sliding_window>]
[output <output_frequency>]

2.3　有状态故障检测机制

有状态错误检测机制(Cesar-FD)由探测器、语句仓储、Cesar 和反馈 4 个子模块构成,如图 3 所示.

(1)探测器:监控目标模块运行状态,创建符合要求的事件实例并将其注入 Cesar 系统,包括不同类型的探测器,如内存探测器、网络探测器等.

(2)语句仓储:存放用户定义的语句,这些语句必须是合法的 Cesar 语句,用来描述故障情景,管理员可以在线添加新的语句.

(3)Cesar:有状态故障检测机制的核心,负责在线处理大量的输入事件流,从中抽取事件序列并触发相应的反馈操作.

(4)反馈:用户自定义的补偿动作,当满足语句描述的故障情况发生时,由 Cesar 负责触发执行.

管理员通过添加 Cesar-FD 规则的方式声明故障场景,同时定义故障场景下的补偿动作.一个 Cesar-FD 规则包含<事件集、语句、反馈>3 部分元素,事件集定义所有涉及到的事件类型,语句定义触发条件,反馈定义补偿动作.用户可以根据运行经验不断添加新的故障场景,逐步提高系统的可靠性.

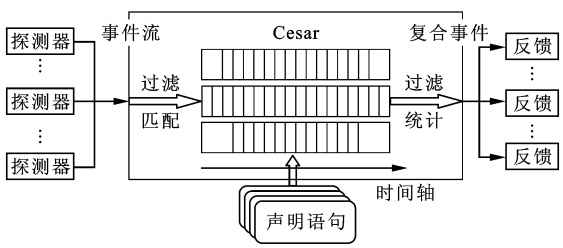


图 3　基于 Cesar 的有状态故障检测机制

3　系统评估

系统评估工作主要从复杂事件处理引擎性能、有状态故障检测机制性能和投机执行机制 3 个方面展开.

3.1　Cesar 引擎性能评测

为了避免网络延迟对实验的影响,事件流将直接注入到 Cesar 引擎中.

我们采用事件实例来描述某一时刻系统的状态变化,采用事件序列来表示系统中几种状态变化依次发生.由于不考虑网络延迟,因此影响性能的因素主要包括事件序列长度和滑动窗口长度.本节将分别评估这 2 个参数对引擎性能的影响.

(1)测试事件序列长度对性能的影响.假设滑动窗口数为 1,长度为 30 s,测试 5 种事件序列:事件序列 2(B,C),事件序列 2-neg(! B,C),事件序列 3(A,B,C),事件序列 3-neg(A,! B,C),事件序列 4(A,B,C,D),它们包含的事件实例个数依次为 2、1、3、2 及 4.

图 4 显示了事件序列长度对性能的影响(这里采用事件序列的长度来衡量事件序列的复杂性).输出事件生成的前提条件是在输入事件流中出现指定的事件序列,假设 Cesar 本身没有任何性能损耗,则输入输出事件的速率比值应该等于事件序列所包含的事件实例数.

由图 4 可以看出:当输入事件速率小于 20 ms^{-1} 时,斜率的变化基本上呈线性;当输入事件速率大于

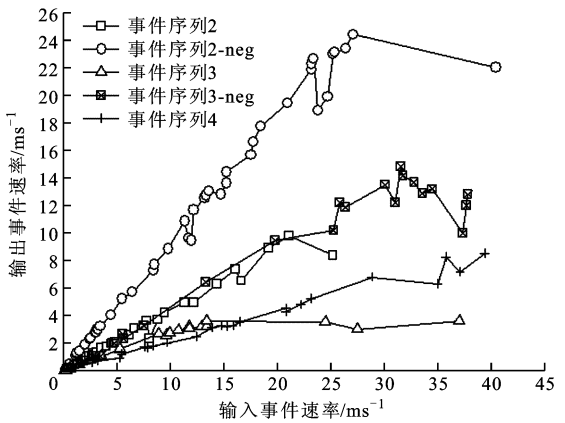


图 4　事件序列长度对引擎性能的影响

20 ms^{-1}时，斜率变化呈非线性.

(2)测试滑动窗口长度对性能影响. 假设事件序列为A,B,C,滑动窗口的数目为1,测试5种不同的滑动窗口长度:10,20,30,60,120 s.

图5所示的是滑动窗口长度对Cesar性能的影响. Cesar处理持续事件流过程中,滑动窗口的大小会影响引擎处理事件的个数,所以滑动窗口的变化可以有效衡量Cesar内部基于NFA的事件序列识别性能. 由图5可以看出,在不同滑动窗口长度的情况下,曲线斜率没有显著变化,这表明经过优化的NFA识别技术具有较好的性能.

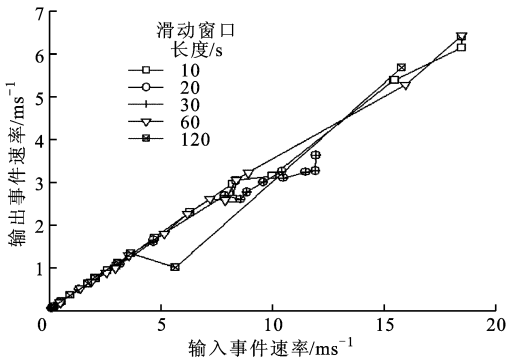


图5 滑动窗口长度对引擎性能影响

由上面2组实验可以发现,Cesar在输入事件速率低于20 ms^{-1}的情况下,能够保持稳定的处理能力.

3.2 有状态故障检测机制性能评测

有状态故障检测机制可以通过3种指标进行衡量:①检测延迟D,即从故障发生到系统检测到错误之间的时间间隔;②误报率,即在未出现故障的情况下发出的报警通告的概率;③漏报率,即在出现故障的情况下未发出报警通告的概率.

误报率和漏报率主要受到几个因素的影响:①事件序列的复杂度(目前通过长度来衡量);②输入事件流的到达速率;③时间/长度窗口个数及长度. 目前还没有公认的测试集合来测试复杂事件流引擎的上述2个指标. Cesar引擎采用的是精确匹配的原则,根据我们实际运行药物设计网格的经验,能够满足实际需要.

检测延迟主要受2个因素影响:①网络传输延迟;②Cesar引擎处理延迟. 由于网络传输延迟受其他因素影响较大,因此我们将输入事件流直接注入到Cesar引擎中. 本文采用一个最为简单的场景来评测Cesar本身导致的延迟,假设事件序列个数为1,长度为2(A,B),滑动窗口格式为1,长度为30 s.

图6显示了实际故障检测延迟与理论值之间的差异. 当输入事件速率低于20 ms^{-1}时,2条曲线之间拟合程度非常高,这表明基于管道机制构建的Cesar引擎自身导致的额外开销非常小.

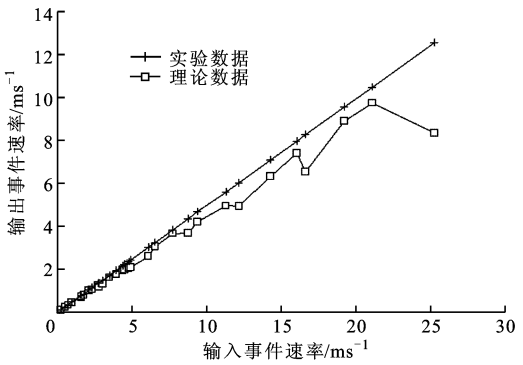


图6 实际故障检测延迟和理论值之间的差异

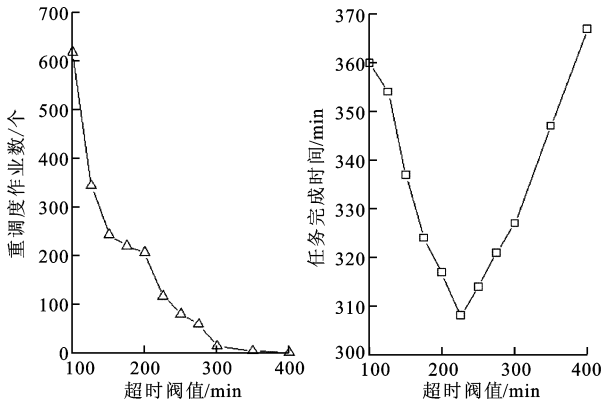
3.3 投机执行机制评测

投机执行机制测试环境包含48个CPU,由位于中科院上海药物所以及北京航空航天大学的3个节点组成. 测试任务采用具有50 000个靶标的CNDP数据库,分子对接工具使用GAsDock软件,每100个分子被切分为一个作业,单个作业处理时间为20 min左右. 理想情况下,任务完成时间为:

(作业量 / 可用资源数) × 单个作业处理时间 (1)

根据式(1),在测试环境下,任务的理论完成时间为240 min.

超时阀值和任务完成时间以及重调度作业数之间的关系如图7所示. 随着超时阀值增加,重调度作业数不断减少. 在起始阶段,任务完成时间由于重调度作业数目减少而缩短,但底层资源处理能力的差异以及资源负载的变化使得任务完成时间又受限于最慢资源的执行时间,进而导致任务完成时间延长.



(a)重调度作业数 (b)任务完成时间

图7 超时阀值对投机执行机制的影响

图 7 表明,通过选取合理的超时阀值可以缩短作业完成时间. 由于式(1)中 3 个指标值随时间变化,控制节点通过收集任务分发和状态通告事件来动态调整超时阀值. 我们注册 2 条规则到 Cesar-FD 规则集中,新注册的规则负责触发反馈操作来动态调整节点的超时阀值.

规则 1　超时阀值调整规则. 运行时动态调整针对某个节点的超时阀值.

(1)事件定义:event*A* 代表作业分发操作,event*B* 代表作业完成通告,由控制节点的探测器来采集.

(2)Cesar 语句:如果作业在 0.5 h 内成功返回,则触发通告操作

select * usepattern seq (event*A*, event*B*) where Event*A*. id=event*B*. id and event*B*. status=finished with win: timein (30 min)

(3)反馈操作:计算某种类型作业在某个节点的平均执行时间,反馈给工作流引擎.

规则 2　重调度规则. 如果作业没有在超时阀值内结束(成功或失败),则执行作业重调度操作.

(1)事件定义:event*A* 代表作业分发操作,event*B*代表作业状态通告,由控制节点的探测器来采集.

(2)Cesar 语句:如果在 0.5 h 内作业没有成功返回,则触发重提交操作

select * usepattern seq (event*A*,! event*B*) where event*A*. id=event*B*. id and event*B*. status=failed or event*B*. status=finished with win: timeout (30 min)

(3)反馈操作:通知工作流引擎执行作业重调度操作.

4　总　结

本文的药物设计网格着重解决了高通量药物虚拟筛选过程中对容错的需求,通过扩展现有框架,依托中国国家网格及自有资源,构建了一个分布的、支持容错的计算环境. 实验表明,本文的措施较好地满足了药物虚拟筛选的实际需求,有效地提高了药物设计网格的稳定性和可靠性.

参考文献:

[1] QIAN Depei. CNGrid: a test-bed for grid technologies in China[C]// Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems. Piscataway, NJ, USA: IEEE, 2004:135-139.

[2] 卢锡城,王怀民,王戟. 虚拟计算环境 iVCE:概念与体系结构 [J]. 中国科学(E 辑:信息科学),2006,36(10):1081-1099.
LU Xicheng, WANG Huaimin, WANG Ji. Virtual computing environment iVCE: concept and architecture [J]. Science in China (Series E: Information Sciences), 2006, 36(10):1081-1099.

[3] LI Honglin, LI Chunlian, GUI Chunshan, et al. GAsDock: a new approach for rapid flexible docking based on an improved multi-population genetic algorithm [J]. Bioorg Med Chem Lett, 2004, 14(18): 4671-4676.

[4] SAHNI S. Scheduling master-slave multiprocessor systems [J]. IEEE Transactions on Computers, 1996, 45(10): 1195-1199.

[5] GUNJAN K, IGNACIO L, FAHAD A, et al. Stateful detection in high throughput distributed systems [C]// Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems. Piscataway, NJ, USA: IEEE, 2007: 275-287.

[6] 孟小峰,周龙骧,王珊. 数据库技术发展趋势[J]. 软件学报,2004,15(12):1822-1836.
MENG Xiaofeng, ZHOU Longxiang, WANG Shan. State of the art and trends in database research [J]. Journal of Software, 2004, 15(12): 1822-1836.

[7] 金澈清,钱卫宁,周傲英. 流数据分析与管理综述[J]. 软件学报,2004,15(08):1172-1181.
JIN Cheqing, QIAN Weining, ZHOU Aoying. Analysis and management of streaming data: a survey [J]. Journal of Software, 2004, 15(8): 1172-1181.

[8] WU E, DIAO Yanlei, SHARIQ R. High-performance complex event processing over streams [C]// Proceeding of the 25th ACM SIGMOD International Conference on Management of Data/Principles of Database Systems. New York, USA: ACM, 2006: 407-418.

[9] WALKER W, MIREK R, JOHANNES G, el at. What is "next" in event processing? [C]// Proceeding of the 26th ACM SIGMOD International Conference on Management of Data/Principles of Database Systems. New York, USA: ACM, 2007: 263-272.

[10] BRUNO N, KOUDAS N, SRIVASTAVA D. Holistic twig joins: optimal XML pattern matching [C]// Proceeding of the 21st ACM SIGMOD International Conference on Management of Data/Principles of Database Systems. New York, USA: ACM, 2002: 310-321.

(编辑　刘杨　赵大良)