基于推测的无响应任务自适应容错调度算法

崔云飞 吴晓进 戴 晔 程 肖 郭 岗
(北京航天飞行控制中心 北京 100094)

摘 要 已有的基于静态的执行失败判定时间阈值的无响应任务容错调度算法,不能适应大数据处理中心动态的集群负载。针对该问题,提出判定无响应任务执行失败时间阈值自适应调整方法。基于该模型,设计了自适应的无响应任务容错调度算法(AFTS)。AFTS 算法通过分析作业规模、单个任务大小和剩余作业推测执行时间等参数,自适应地调整无响应任务判定执行失败的时间阈值,以减少无响应任务对整体作业执行效率的影响,降低作业响应时间。基于开发的原型系统,验证了自适应判定方法,测试了算法的性能。实验结果表明,AFTS 算法在作业响应时间等方面优于已有的无响应任务容错调度算法。

关键词 大数据,容错调度,自适应,推测的,MapReduce
中图法分类号 TP393 **文献标识码** A

Adaptive Fault-tolerant Scheduling Algorithm for Unresponsive Task Based on Speculation

CUI Yun-fei WU Xiao-jin DAI Ye CHENG Xiao GUO Gang
(Beijing Aerospace Control Center, Beijing 100094, China)

Abstract Current fault-tolerant scheduling algorithm for unresponsive task, based on static execution failed time threshold, can not adapt to dynamic cluster load of large data processing center. To address this issue, an adaptive execution failed time threshold method was proposed. Based on this method, an adaptive fault-tolerant scheduling algorithm (AFTS) for unresponsive task was designed. AFTS adjusts unresponsive task's time threshold to be determined failure dynamically and to reduce the job response time, according to the information of job size, the size of individual tasks and the remaining operating time. A prototype system using AFTS is developed, on which the validation of the adaptive execution failed time threshold method and the evaluation of AFTS's performance are carried out. It is shown that AFTS outperforms current fault-tolerant scheduling algorithm in term of the job response time.

Keywords Big data, Fault-tolerant scheduling algorithm, Adaptive, Speculative, MapReduce

1 引言

随着科学技术的快速发展,科学研究、互联网服务、电子商务等多领域均呈现出数据量激增的趋势,如何对大数据进行高效处理成为亟需解决的问题。为了应对大数据处理的挑战,Google 提出了 MapReduce 计算模型[1,2],利用 Apache Hadoop 对 MapReduce 实现了开源,该计算模型已经成为目前最流行的大数据处理工具。MapReduce 目前有两个版本:使用最广泛的 MR1 和正在测试的 Yarn,它们主要的框架均是设定系统由主控节点(Master)和数据节点(Slave)组成。Master 负责接收用户的请求 job 和管理整个集群的 Slave,首先将 job 拆分成多个 task,并根据 Slave 的负载情况分配 task;然后对 Slave 节点信息和运行 task 进行状态监控,并根据设定的容错调度策略进行动态调度调整;最后汇总 Slave 完成的 task 信息,给出 job 处理结果。

在异构环境中,由于机架、硬盘、CPU、Memory,甚至是供电等多个方面都可能出现故障,而且在数据密集型应用环境中,失效成为了常态。因此,在每一次 job 执行过程中,当部分节点出现故障等情况时,考虑如何来进行即容错调度成为一个亟需解决的问题。在 MapReduce 框架下,Slave 需要定时通过心跳等机制向 Master 汇报节点状态和在其上运行的 task 的执行状态。当 task 长时间无法完成,甚至是长时间没有响应时,Master 需要对其自动识别并进行自行恢复。当任务执行缓慢导致长时间无法完成时,需要采用推测执行(Speculative Task)等[3-6]机制进行自行恢复;当 task 长时间没有响应时,需要进行及时的容错调度进行解决[7-10]。目前针对 task 无响应的容错调度机制,需要在系统运行前,进行参数配置,设置一个常量阈值,当 task 无响应的时间达到该阈值时,Master 中的 JobTracker 将判定该 task 出现故障,fail 掉该 task,并在另外一个计算节点上重新执行该 task。但是

本文受部委级重点项目资助。
崔云飞(1986-),男,博士,工程师,主要研究方向为大数据处理、资源调度,E-mail:sducyf123@aliyun.com;吴晓进(1973-),男,高级工程师,主要研究方向为数据处理,E-mail:wuxiaojin@aliyun.com;戴 晔(1975-),女,高级工程师,主要研究方向为数据处理,E-mail:daiye@aliyun.com;程 肖(1983-),男,工程师,主要研究方向为数据处理,E-mail:chengxiao@aliyun.com;郭 岗(1979-),男,工程师,主要研究方向为数据处理,E-mail:guogang@aliyun.com。

在数据密集型应用中，集群环境和作业规模都在不断变化，静态无响应任务判定阈值不能适应这种动态变化。

在本文中，针对集群环境和作业规模的动态性，提出一种无响应任务自适应容错调度算法。该算法通过实时分析不同job中task的平均执行速度与数据处理资源的规模和负载，动态调整判定无响应任务执行失败的时间阈值，从而在保证资源利用率的同时，提高集群的容错能力并降低无响应任务对整个作业执行时间的影响。

2 研究背景

2.1 Hadoop作业调度流程及执行描述

Apache Hadoop的MapReduce框架是基于Google MapReduce原理实现的开源软件，是目前最流行的大数据处理工具。

Hadoop的MapReduce框架的基本体系结构如图1所示，主要由JobTracker和TaskTracker两类基本部件组成。目前Hadoop Yarn框架对MapReduce框架进行了升级，增加了应用管理器ApplicationManager，降低了中心节点的调度压力，但是在进行任务级细粒度分配时的原理与MapReduce框架中JobTracker和TaskTracker的原理类似。

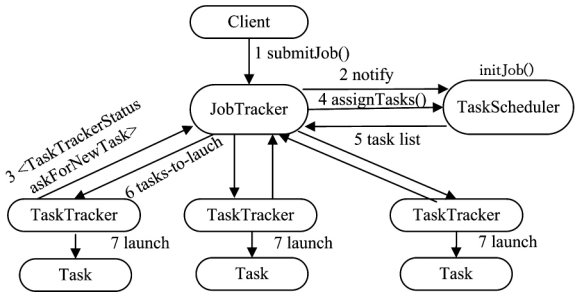

图1 Hadoop MapReduce框架调度流程

JobTracker为中心控制节点，主要处理任务的分配和调度，记录任务运行状态和已分配任务的完成情况，处理TaskTracker出现的异常现象等。如果某一个TaskTracker控制的任务被判定为执行失败的无响应任务，JobTracker会通过无响应任务处理机制选择一个空闲的TaskTracker启动该任务，并将该无响应任务fail掉。

TaskTracker响应JobTracker的控制命令，并按照指令的要求处理数据。TaskTracker监控任务执行，并通过心跳机制定期向JobTracker汇报任务执行状态。TaskTracker分为两类：Mapper和Reducer。Mapper和Reducer可以在任意一个TaskTracker上执行，而JobTracker根据TaskTracker的负载情况向Mapper或者Reducer分配任务。

图1中的步骤1—步骤7表示Hadoop的MapReduce框架的调度流程，其中主要包括提交作业、作业初始化、任务请求、任务调度、任务分配等。

当Hadoop的MapReduce框架执行作业时，单个作业job被拆分成多个任务task执行。由JobInProgress监控job的执行进度，TaskInProgress监控单个task的执行，task的执行采用task attempt机制。正常情况下，每个task启动一个task attempt，当task attempt被判定为慢任务后，控制中心会选择一个合适的节点为对应的慢任务再启动一个task attempt，称为备份任务，这两个task attempt同时运行，哪个先执行完，就采用哪个的结果，并kill掉另一个task attempt，killed task可能被再次调度到任何一个节点上（包括曾经失败过的节点）。当检测到任务长时间无响应或确定任务执行失败后，控制中心会为该任务启动一个相同的task attempt，并fail掉无响应的task attempt，使failed task再次调度时不会在那些曾经失败的节点上运行。

Hadoop MapReduce作业描述方式如图2所示。

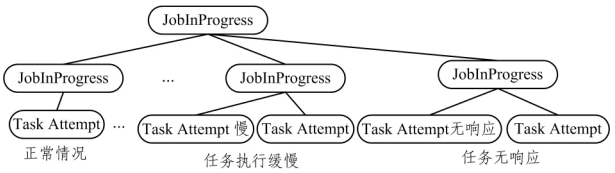

图2 Hadoop MapReduce作业描述方式

2.2 问题的提出

为了降低无响应任务对作业整体执行效率的影响，Hadoop的各个版本中的MapReduce框架均设置了一定的算法来应对无响应任务的容错调度。无响应任务的处理有两种情况：1）设置尝试次数（attempts），超过尝试次数判定为失败，重新选择一个tasktracker执行；2）整个tasktracker不汇报心跳，导致任务无响应，超过一定的时限后，判定该tasktracker失效，将其从调度域中剔除，并将其上执行的任务重新执行。

设置尝试次数比较简单有效，本文不做改进，而整个tasktracker不汇报心跳从而导致任务无响应的情况，在数据密集型环境中比较常见，需要进一步地优化这种情况下的无响应任务容错调度机制，来降低各种故障造成的任务无响应对整个作业响应效率的影响。一个tasktracker可能同时执行一个或者多个任务，而多个任务的响应情况会有差异，因此，本文将从更细粒度（以单个任务为对象）来研究无响应任务的容错调度机制。

无响应任务判定执行失败的时间阈值是影响MapReduce框架容错调度的一个关键因素。过长的无响应任务判定执行失败的时间阈值将造成额外的时间开销，而过短的无响应任务判定执行失败的时间阈值将会增加误判的概率，从而造成资源的浪费。在目前的Hadoop系列版本中，无响应任务判定执行失败的时间阈值是一个常量，用户需要预先在配置文件中设定好无响应任务判定执行失败的时间阈值。

然而数据密集型处理系统是动态变化的，每个job的task需要的处理时间也是不同的。因此，静态的无响应任务判定执行失败的时间阈值将不能适应这种动态变化，有必要针对数据密集型处理系统的集群动态性和作业规模的变化研究无响应任务的自适应判定执行失败的时间阈值，进行无响应任务的自适应容错调度。

3 无响应任务自适应容错调度算法

本文提出的无响应任务自适应容错调度算法的基本思想是：当一个job被拆分成多个task并调度到计算节点上后，首先，计算一个job拆分成task的数量得到作业规模；其次，监控各个task的执行进度和执行时间，推测出单个task实时的平均执行速度；再次，根据数据处理资源规模推测出整个job的剩余执行时间；然后，根据无响应任务容错调度的自适应方法以及实时的状态信息，计算判定无响应任务执行失败的时间阈值；最后，比较无响应任务已经调度执行的时间，做出容错调度策略。

3.1 问题定义

定义 1(无响应任务) 无响应任务是指在某个时间阈值内没有向 JobTracker 发送响应信息的 task。超过一定的时间阈值,Hadoop 中的 MapReduce 机制将 fail 掉该 task,并在其它节点上重新执行该 task。

定义 2(无响应任务判定失败时间阈值) 无响应任务判定失败的时间阈值记为 T^N。JobTracker 在 T^N 时间内等待接收任务 $task_i$ 的响应信息,若超过 T^N 没有收到 $task_i$ 的响应信息,则将 $task_i$ 判定为执行失败的无响应任务。

定义 3(数据处理资源规模) 数据处理资源规模是指控制中心可以调度给 job 的计算节点(slave)的规模。对于一个具体的 job,数据处理资源规模是指,控制中心调度给该 job 的数据处理资源量,记作 n。

定义 4(作业规模) 作业规模是指,一个作业 job 被拆分成任务 task 后的规模。在 Hadoop 的 MapReduce 框架下,单个 job 会被拆分成多个 task,每个 task 执行一个数据块。因为 MapReduce 所需的数据是按照固定大小进行分块存储在分布式文件存储系统中的,所以具体的 job 被拆分后的大部分 task 执行的数据量大小相同,只有 1 个 task 执行的数据量大小不同(原因是分块存储时,按照同样大小进行划分后剩余的数据量小于数据块的划分阈值)。因此,一个 job 的作业规模与 job 被拆分的 task 的数量(记作 m)成正比,也与单个 task 需要处理的数据量(记作 d)成正比。

定义 5(剩余作业规模) 剩余作业规模是指,在一个 job 执行过程中,所有还没有执行成功的 task 的规模。剩余作业规模同剩余 task 的数量以及单个 task 需要处理的数据量成正比。

定义 6(单个任务执行速度) 一个 job 中的 $task_i$ 在数据处理节点 j 上的执行速度(ProgressRate,记为 pr_{ij}),可以直接使用执行进度(ProgressScore,记为 ps_{ij})和执行时间来表征(deltaTime,记为 t_{ij}),其中 ProgressScore 和 deltaTime 可以通过 MapReduce 的原有机制直接获得,且 $ps=$ ProgressScore$\in[0,1]$。如果 $ps=0$,将对后面的除法运算产生影响,为了后续的运算,本文将任务还没有执行时的执行进度设置为 $ps=0.001$。

$$pr_{ij}=ps_{ij}/t_{ij} \tag{1}$$

因为具体的一个 job 中存在很多相同的 $task$(每个 $task$ 处理的数据量相同),所以可以使用 $task_i$ 的平均执行速度表示整个 job 中的相同 $task$ 的平均执行速度。$task_i$ 在不同数据处理节点上的执行进度和执行时间,可以使用与 $task_i$ 相同的 $task$ 的执行进度和执行时间来表示。因此 job 中的 $task_i$ 在调度的数据处理集群中的平均执行速度 pr_i,可由式(1)推算得到:

$$pr_i=\frac{1}{S_n}\sum_{k=1}^{n}(\alpha_k\times pr_{ik})=\frac{1}{S_n}\sum_{k=1}^{n}(\alpha_k\times\frac{ps_{ik}}{t_{ik}}) \tag{2}$$

$$S_n=\sum_{k=1}^{n}\alpha_k \tag{3}$$

其中,n 表示执行该 job 的数据处理节点(Slave)的数量,α_k 表示第 k 个 Slave 上可以同时执行的 task 数量,S_n 表示分配给 job 的 n 个 Slave 中任务槽(Slot)的个数。

定义 7(作业剩余执行时间) 作业剩余执行时间包括两类任务的可推测的剩余执行时间,一类是还没有被分配执行的任务的推测执行时间,另一类是正在执行但还没有执行成功的任务的推测执行时间。用 T_r^J 表示作业剩余执行时间,用 m_r 表示还没有被分配的任务的数量,用 T_r^m 表示还没有被分配的任务的推测执行时间。因为分配给该 job 的 Slave 的 slot 的数量为 S_n,所以用 S_n 表示正在执行的任务的数量(如果某个 slot 上空闲,则假设存在一个虚拟任务,该任务的剩余执行时间为零),用 T_r^s 表示正在执行的所有任务可推测的剩余执行时间之和。则:

$$T_r^J=T_r^m+T_r^s=\sum_{i=1}^{m_r}\frac{1}{pr_i}+\sum_{x=1}^{S_n}AET_x \tag{4}$$

其中,AET_x 表示正在执行的任务 $task_x$ 可推测的剩余执行时间。

$$AET_x=\frac{1-ProgressScore}{ProgressRate}=\frac{1-ps_x}{pr_x}=\frac{1-ps_x}{ps_x}\times t_x \tag{5}$$

3.2 无响应任务被判定执行失败时间阈值调整模型

在节点 j 上执行的无响应任务 $task_i$ 被判定为执行失败的时间阈值 T_{ij}^N 需要大于单个 $task$ 在节点 j 上执行的时间,即

$$T_{ij}^N>\frac{1}{pr_{ij}} \tag{6}$$

但是,无响应任务的执行进度 $ps_i=0$,需要使用平均执行速度进行计算,故需要使用 $task_i$ 在集群中的平均判定执行失败的时间阈值 T_i^N 表示,则:

$$T_i^N>\frac{1}{pr_i}=\frac{1}{\frac{1}{S_n}\sum_{k=1}^{n}(\alpha_k\times pr_{ki})} \tag{7}$$

综上,本文设置无响应任务 $task_i$ 被判定为执行失败的时间阈值 T_i^N 的表达式为:

$$T_i^N=\begin{cases}1\text{min}, & \min(T_r^J,3\times\frac{1}{pr_i})<1\text{min}\\ \frac{1}{2}\times(\frac{1}{pr_i}+\min(T_r^J,3\times\frac{1}{pr_i})), & \min(T_r^J,3\times\frac{1}{pr_i})\geqslant 1\text{min},\ \text{且 } T_r^J>\frac{1}{pr_i}\\ \frac{1}{pr_i}, & T_r^J\leqslant\frac{1}{pr_i}\end{cases} \tag{8}$$

3.3 AFTS 算法描述

无响应任务自适应容错调度(AFTS)算法由计算单个任务平均执行速度、计算整体作业剩余执行时间、调整任务执行失败判定时间阈值以及判断任务执行是否失败 4 个阶段组成。通过单任务推测算法计算单个任务平均执行速度;通过作业剩余执行时间推测算法计算整体作业剩余执行时间;通过无响应任务执行失败的时间阈值调整模型求解得到无响应任务被判定执行失败的时间阈值;通过比较任务已经调度但无响应的时间和无响应任务被判定执行失败的时间阈值,确定是否判定该无响应任务执行失败,是否需要进行容错调度并且重新将该任务调度到一个新的节点执行。因为 AFTS 算法可以动态调整无响应任务被判定为执行失败的时间阈值,所以在整体作业剩余执行时间较短时,AFTS 算法减少了不必要的过长判定时间等待,提高了作业的执行性能;在整体作业剩余执行时间较长时,AFTS 算法延长无响应任务被判定为执行失败的时间阈值,避免了不必要的误判,提高了资源利用率,保证了作业的整体执行性能。

AFTS 算法的流程如算法 1 所示。

算法 1　无响应任务自适应容错调度算法

输入：无响应任务 $task_i$ 已经调度到数据处理节点的时间记为 T_i^A。

输出：无响应任务 $task_i$ 是否判定为执行失败，若执行失败，重新调度 $task_i$ 到另一个节点执行。

统计正在执行的 task 的开始时间和当前时间，计算出已经调度到数据处理节点的时间，将正在执行的 task 按照已经调度到数据处理节点的时间从大到小排序，并将排序结果保存到 dealingtasks 中。

```
Begin
  for task_i in dealingtasks do
      pr_i←gettaskspeed()
      //获取单个任务 task_i 的平均执行速度
      T_r^J←getremanenttime()
      //获取作业剩余执行时间
      T_i^N←getThreshhold()
      //计算无响应任务 task_i 被判定执行失败的时间阈值 T_i^N
      if T_i^A<T_i^N then
        continue;
      else
        判定 task_i 执行失败;
          将 task_i 调度到一个新的节点重新执行;
          return;
        end if
  end for
  return;
End
```

4　实验与结果分析

为了分析文中提出的无响应任务自适应容错调度算法(AFTS)的性能，下面将 AFTS 算法和 Hadoop 中原始的无响应任务容错调度算法(FTS)进行比较。通过在实验集群的实验，分析算法的性能。

4.1　实验环境

本节主要描述实验的环境，以及环境配置的各个参数。使用实验室的 16 台 PC 机进行实验集群的搭建，各 PC 机采用 1000M 的局域网互联。这 16 台 PC 机是异构的，如表 1 所列。

表 1　实验集群环境配置

机器	机器数	操作系统	磁盘	CPU	内存
Lenovo	6	Centos 6.3	500GB	2	2GB
Lenovo	4	Centos 6.3	500GB	2	1GB
Dell	6	Centos 6.3	500GB	2	2GB

原型系统基于 Hadoop-0.23.5 开发，Master 模块部署于管理节点，Slave 模块部署于 15 个计算节点，1000Mbit/s 以太网作为数据传输网络。输入文件由 Hadoop 分布式文件系统管理，文件块存储于计算节点的本地硬盘，每个文件块的大小为 100K 至 64M(用于测试无响应任务判定时间阈值对处理不同大小数据块的影响)，并且有两个副本。每一个 MapReduce job 作为一个作业，而作业中的 Map Task 作为任务。实验利用 Hadoop 自带的配置文件设置无响应任务判定时间阈值，利用 Adaptive Capacity Scheduler 模块实现 AFTS 算法。

4.2　实验设置

为了比较调度算法对不同规模作业的影响，实验依照单个 task 处理的数据量分成 5 组，分别为 100K、1M、10M、32M 和 64M。每组测试的任务数均取 1000 个任务，具体设置如表 2 所列。

表 2　作业参数设置

组号	单任务处理数据量	任务数	数据量
1	100K	1000	100MB
2	1M	1000	1000MB
3	10M	1000	10GB
4	32M	1000	32GB
5	64M	1000	128GB

通过提交这 5 组不同的作业，分析单个任务完成时间和 AFTS 算法的自适应性，以及比较各容错调度算法对作业响应时间的影响。

4.3　单个任务平均执行时间分析

向 Hadoop 提交 4.2 节设置中的第 5 组作业，通过 Hadoop 自带的任务状态监控机制，直接获取每个正常任务的执行时间，然后对相同数据量的任务的执行时间取平均值，得到不同数据量下的单个任务的平均执行时间，如图 3 所示。

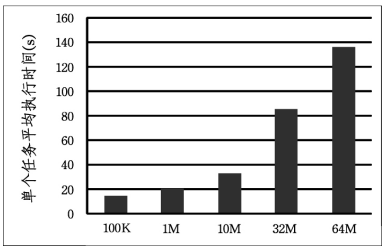

图 3　单个任务平均执行时间

从图 3 中看出，随着需要处理的数据量的增加，单个任务的执行时间也在增加，同时也说明我们在进行无响应任务容错调度时需要根据不同大小的任务动态地调整判定执行失败的时间阈值。另外，单个任务的平均执行时间与数据量的大小并非呈严格的线性关系，原因是，在集群处理过程中采用心跳等机制会占用一定的时间开销。

4.3.1　作业响应时间分析

为了对比基于推测的无响应任务容错调度算法(AFTS)和原有的无响应任务容错调度算法(FTS)，向不同算法下的系统提交 4.2 节中实验设置的 5 组作业时，针对相同组作业执行过程，制造相同的故障场景，来验证不同算法对作业响应时间的影响。不同算法下的作业响应时间对比如图 4 所示，图 4 直观地显示出 AFTS 降低作业响应的效果。客观地反映出在作业规模变化过程中，自适应容错调度算法可以有效地降低作业的整体响应时间。

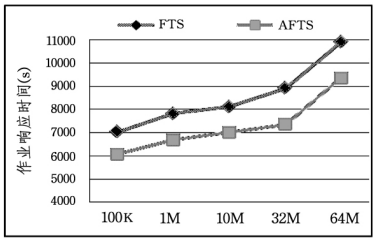

图 4　不同算法下作业响应时间对比

结束语　本文提出了一种基于集群规模/作业剩余规模推测的无响应任务自适应容错调度算法。该算法与已有的无响应任务容错调度算法相比，不同点在于不再采用静态的判定执行失败的时间阈值，而是通过实时推测单个任务的平均

执行速度和作业剩余执行时间，对无响应任务判定执行失败的时间阈值进行自适应调整，从而降低了无响应任务对作业响应时间的影响。在下一步的工作中，将研究怎样同时处理无响应任务和慢任务对作业响应时间的影响。

参 考 文 献

[1] Dean J，Ghemawat S. MapReduce：simplified data processing on large clusters [J]. Communications of the ACM，2008，51(1)：107-113

[2] 陆嘉恒. Hadoop 实战[M]. 机械工业出版社，2012

[3] Adaptive Scheduler[EB/OL]. https://issues. apache. org/jira/browse/MAPREDUCE-1380，2013

[4] Improve speculative execution [EB/OL]. https://issues. apache. org/jira/browse/MAPREDUCE-2039，2013

[5] Speculative execution for Reads [EB/OL]. https://issues. apache. org/jira/browse/CASSANSRA-4705，2013

[6] Looking for speculative tasks is very expensive [EB/OL]. https://issues. apache. org/jira/browse/MAPREDUCE-4499，2013

[7] Dinu F，Ng T S E. Understanding the Effects and Implications of Compute Node Related Failures in Hadoop[R]. HPDC'12. 2012：18-22

[8] Lee K H，Lee Y J，Choi H，et al. Parallel Data Processing with MapReduce：A Survey[J]. SIGMOD Record，2011，40(4)：11-20

[9] Matei Z，Andy K，Anthony D. Improving MapReduce Performance in Heterogeneous Environments[C]// 8th Usenix Symposium on Operating Systems Design and Implementation. 2008

[10] ResourceManagerRest [EB/OL]. http://hadoop. apace. org/docs/r0. 23. 6，2013

(上接第 5 页)

[9] Imig T J，Adria′N H O. Binaural columns in the primary field (A1) of cat auditory cortex[J]. Brain Research，1978，138(2)：241-257

[10] Xin J，Qi Y. Auditory Modeling，in Mathematical Modeling and Signal Processing in Speech and Hearing Sciences[C]// Springer International Publishing：Cham. 2014：67-113

[11] Allen J B. A physical model of cochlear transduction[J]. Journal of the Acoustical Society of America，1980，67(Suppl 1)：S46

[12] Allen J B. Cochlear micromechanics--a physical model of transduction[J]. Journal of the Acoustical Society of America，1980，68(6)：1660-1670

[13] Manley G A，Narins P M，Fay R R. Experiments in comparative hearing：Georg von Békésy and beyond[J]. Hearing Research，2012，293(1/2)：44-50

[14] Lyon R F. A computational model of filtering，detection，and compression in the cochlea[C]// IEEE International Conference on Acoustics. 1982

[15] Goldstein J L. Modeling rapid waveform compression on the basilar membrane as multiple-bandpass-nonlinearity filtering [J]. Hearing Research，1990，49(1-3)：39-60

[16] Meddis R，O'Mard L P，Lopez-Poveda E A. A computational algorithm for computing nonlinear auditory frequency selectivity [J]. Journal of the Acoustical Society of America，2001，109(6)：2852-2861

[17] Sumner C J，et al. A nonlinear filter-bank model of the guinea-pig cochlear nerve：rate responses[J]. Journal of the Acoustical Society of America，2003，113(6)：3264-3274

[18] Sumner C J，et al. A revised model of the inner-hair cell and auditory-nerve complex[J]. Journal of the Acoustical Society of America，2002，111(5)：2178-2188

[19] 李朝晖，迟惠生. 听觉外周计算模型研究进展[J]. 声学学报，2006，31(5)：449-465

[20] Hodgkin A L，Huxley A F. Propagation of electrical signals along giant nerve fibers[J]. Royal Society of London Proceedings，1952，140(899)：177-183

[21] Izhikevich E M. Dynamical Systems in Neuroscience[R]. Riverview Drive Suite Totowa New Jersey，2010

[22] Louis T，Sornborger A T. Dimensionally-reduced visual cortical network model predicts network response and connects system- and cellular-level descriptions [J]. Journal of Computational Neuroscience，2009，28(1)：91-106

[23] Tao L，Lauderdale J D，Sornborger A T. Mapping Functional Connectivity between Neuronal Ensembles with Larval Zebrafish Transgenic for a Ratiometric Calcium Indicator [J]. Frontiers in Neural Circuits，2011，5(4)：2

[24] Diesmann M，Gewaltig M O，Aertsen A. Stable propagation of synchronous spiking in cortical neural networks [J]. Nature，1999，402(6761)：529-533

[25] Sornborger A T，Tao L. A Unified Framework for Information Coding：Oscillations，Memory，and Zombie Modes [J]. Eprint Arxiv，2014，arXiv：1410. 1116

[26] Sornborger A T，Tao L. Exact，Dynamically Routable Current Propagation in Pulse-Gated Synfire Chains [J]. Eprint Arxiv，2014，arXiv：1410. 1115

[27] Wang C，et al. A Fokker-Planck approach to graded information propagation in pulse-gated feedforward neuronal networks[J]. arXiv preprint arXiv：1512. 00520，2015

[28] Rumelhart，et al. Parallel Distributed Processing. Explorations in the Microstructure of Cognition，Vol. I：Foundations [J]. Mit Press Cambridge Ma，1986，63(4)

[29] Horwitz B，Husain F T. Simulation Frameworks for Large-Scale Brain Systems，in Handbook of Brain Connectivity[M]. Pringer Berlin Heidelberg：Berlin，Heidelberg，2007：275-302

[30] Husain F T，et al. Relating neuronal dynamics for auditory object processing to neuroimaging activity：a computational modeling and an fMRI study[J]. Neuroimage，2004，21(4)：1701-1720

[31] Izhikevich E M，Edelman G M. Large-scale model of mammalian thalamocortical systems[J]. Proceedings of the National Academy of Sciences of the United States of America，2008，105(9)：3593-3598

[32] Gewaltig M O，Diesmann M. NEST (NEural Simulation Tool) [J]. Scholarpedia，2007，2(4)

[33] Plesser H E，et al. NEST：the Neural Simulation Tool[M]// Encyclopedia of Computational Neuroscience. Springer New York，2015：1849-1852

[34] Schmidt M，et al. Integrating multi-scale data for a network model of macaque visual cortex[J]. Bmc Neuroscience，2013，14(1)：1-2

[35] Ishii S，Diesmann M，Doya K，et al. Multi-scale，multi -modal neural modeling and simulation[J]. Neural Networks，2011，24(9)：917-917

[36] Mehring C，et al. Activity dynamics and propagation of synchronous spiking in locally connected random networks[J]. Biological Cybernetics，2003，88(5)：395-408