面向分布式系统的复制数据类型理论研究概述

(CCF 2018 第九届优博论坛)

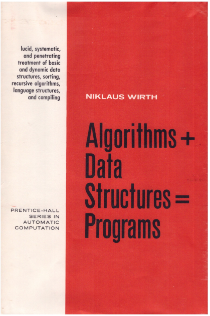
魏恒峰

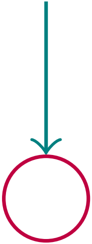
南京大学软件所

2018 年 08 月 08 日

Abstract Data Types (ADT) [Liskov and Zilles, 1974]

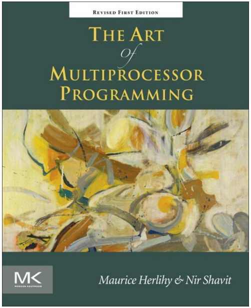
(单线程; 顺序语义)

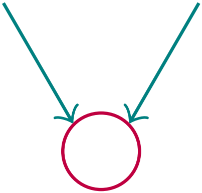




Concurrent Data Types [Herlihy and Wing, 1990]

(多线程; 并发语义)

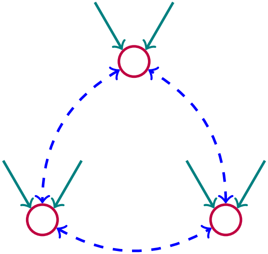




PL (Programming Language)

Replicated Data Types (RDT; ≈ 2010 年) [Burckhardt et al., 2014]

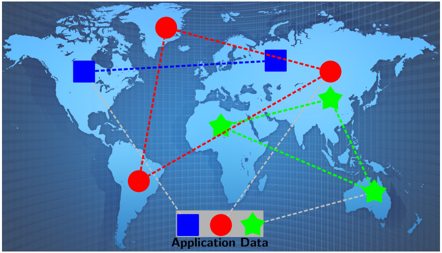
(多副本; 复制语义)



DC (Distributed Computing)

新平台：大规模分布式系统

低延迟　高可用性 (4 个 9)　高容错性　高可扩展性



分布数据 (distributed data):

1. 分区 (partition): 水平扩展
2. 副本 (replication): 就近访问, 容灾备份

复制数据类型 [Shapiro et al., 2011a]

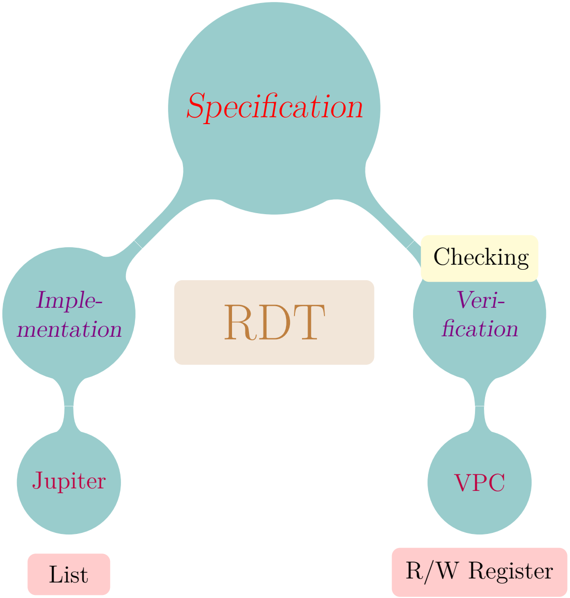
- Read/Write Register
- Counter
- Set
- List
- HashMap
- Disjoint Set
- Graph
- ...

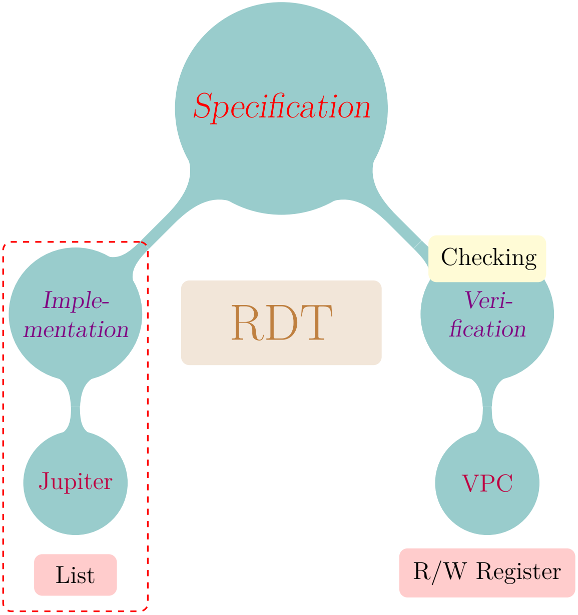
新问题，新挑战

Replicated Data Types: Specification, Verification, Optimality

Sebastian Burckhardt   Alexey Gotsman   Hongseok Yang   Marek Zawirski

[Burckhardt et al., 2014]


Specification
Imple-
mentation
RDT
Checking
Veri-
fication
Jupiter
VPC
List
R/W Register


Specification
Imple-
mentation
RDT
Checking
Veri-
fication
Jupiter
VPC
List
R/W Register

Brief Announcement @ PODC'2018 [1]

实现复制列表的 Jupiter 协议 [Nichols et al., 1995][a] 满足 weak list specification [Attiya et al., 2016][b].

[a] David A. Nichols et al. (1995). "High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System". In: *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*. UIST '95. ACM, pp. 111–120.

[b] Hagit Attiya et al. (2016). "Specification and complexity of collaborative text editing". In: *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*. PODC '16. ACM, pp. 259–268.

[1] 藏在脚注里的猜想@PODC'2016 [Attiya et al., 2016]

Weak List Specification

基于副本的协同文本编辑系统

(a) Google Docs

(b) Apache Wave

(c) Wikipedia

(d) LaTeX Editor

复制列表对象：建模编辑系统的核心功能

$\textsc{Ins}(a, p)$ ： 在 p 位置插入元素 a

$\textsc{Del}(p)$ ： 删除 p 位置上的元素

$\textsc{Read}$ ： 返回该列表

Specification and Complexity of Collaborative Text Editing

Hagit Attiya
Technion

Sebastian Burckhardt
Microsoft Research

Alexey Gotsman
IMDEA Software Institute

Adam Morrison
Technion

Hongseok Yang
University of Oxford

Marek Zawirski*
Inria & Sorbonne Universités,
UPMC Univ Paris 06, LIP6

定义 (Weak List Specification $\mathcal{A}_{\text{weak}}$ [Attiya et al., 2016])

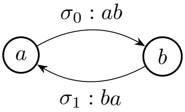
Informally, $\mathcal{A}_{weak}$ requires the ordering between elements that are not deleted to be consistent across the system.

定义在系统所有列表状态上的全局性质

定义（状态对兼容性 (Pairwise State Compatibility Property)）

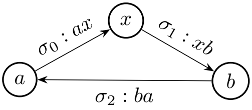
任给两个列表状态 σ_0、σ_1，若它们含有两个共同元素 a、b，则 a、b 在 σ_0 与 σ_1 中的相对顺序保持一致。

$\boxed{\sigma_0 : ab}$ $\boxed{\sigma_1 : ba}$



✗

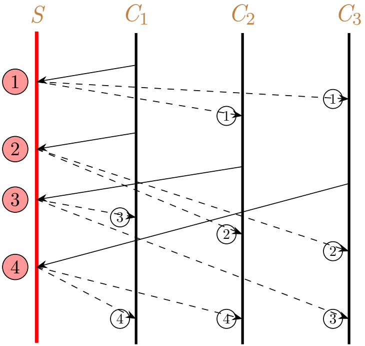
$\boxed{\sigma_0 : ax}$ $\boxed{\sigma_1 : xb}$ $\boxed{\sigma_2 : ba}$



✓

Jupiter

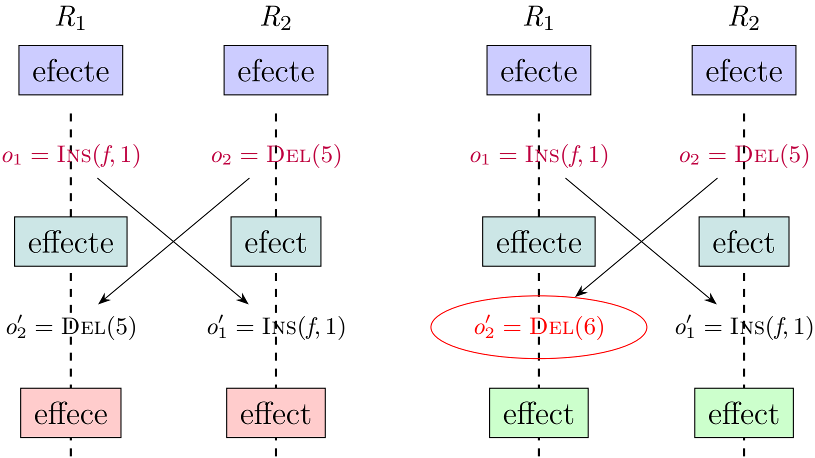
$(n+1)$ replicas $\triangleq$ (n) Client + (1) Server [Nichols et al., 1995]

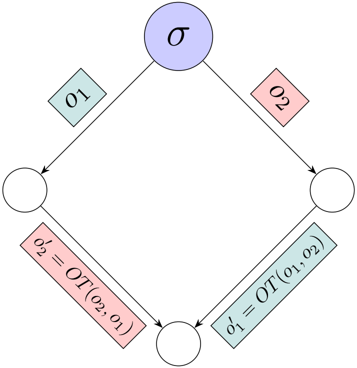


Server 负责将所有操作序列化

用户操作 $\xrightarrow[\text{立即返回}]{\text{提交}}$ Client $\xrightarrow[\text{更新操作}]{\text{FIFO}}$ Server $\xrightarrow[\text{更新操作}]{\text{FIFO}}$ 其它 Clients

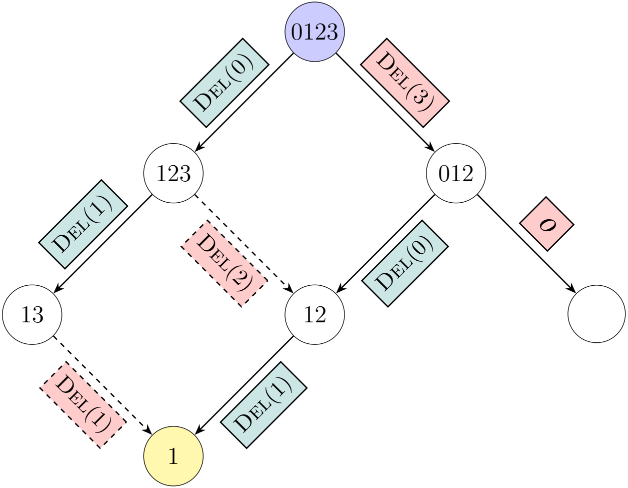
操作转换 (Operational Transformation; OT) [Ellis and Gibbs, 1989]



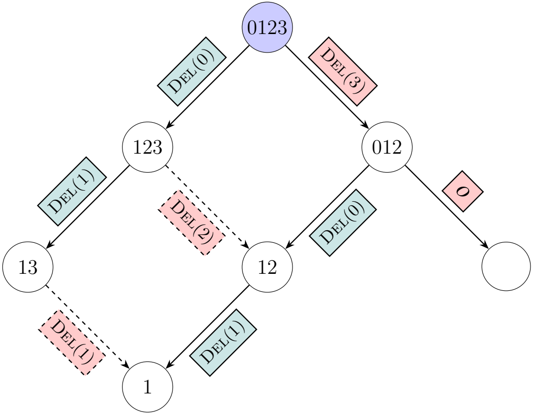


交换律 $\sigma; o_1; o_2' \equiv \sigma; o_2; o_1'$

[Ellis and Gibbs, 1989]

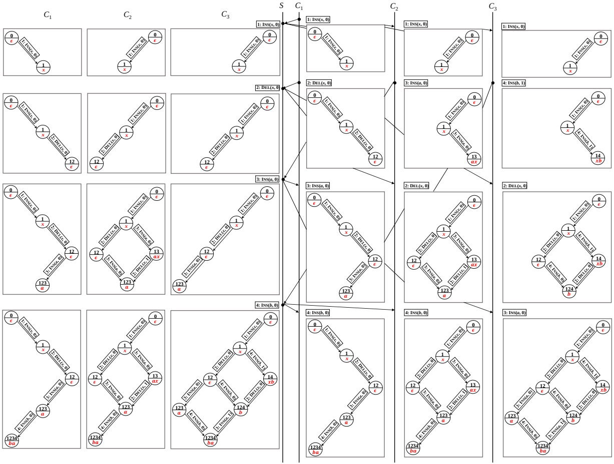

0123
Del(0)
Del(3)
123
012
Del(1)
Del(2)
Del(0)
0
13
12
Del(1)
Del(1)
1

利用数据结构 2D 状态空间 [Xu, Sun, and Li, 2014]
控制何时以及如何执行"操作转换"



2D: Local *vs.* Global

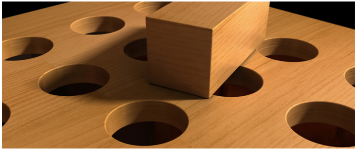
每个 Client 维护一个 $2D$ 状态空间



Server 维护 n 个 $2D$ 状态空间, 与 n 个 Clients 对应

Mismatch!

$\mathcal{A}_{\text{weak}}$ 所规定的全局性质



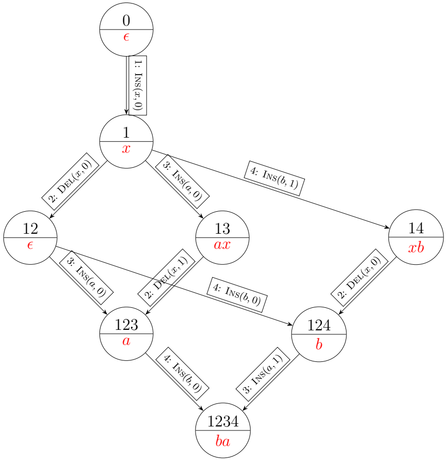
Jupiter 协议中，每个 replica 所维护的局部视图

CJupiter (Compact Jupiter)

定理 (等价性)

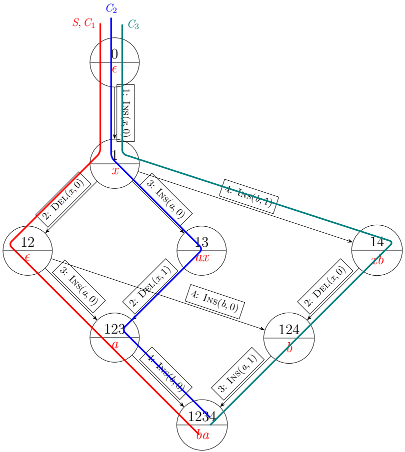
在相同的操作调度下，CJupiter 与 Jupiter 中的对应 *replica* 的行为 *(*状态序列*)* 是相同的。

CJupiter 为每个 replica 维护一个 *n*-ary 有序状态空间



命题 (Compactness of CJupiter)

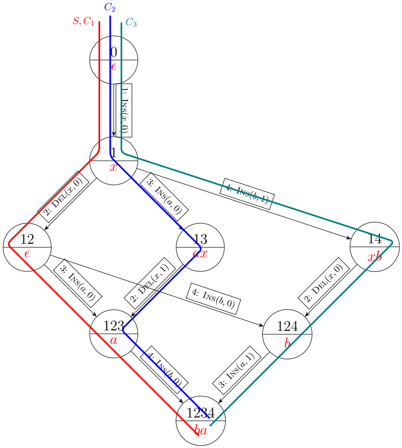
CJupiter 所维护的 $(n+1)$ 个 *n-ary* 有序状态空间是相同的。



每个 replica 的行为对应于该状态空间中的一条路径

CJupiter 满足 Weak List Specification

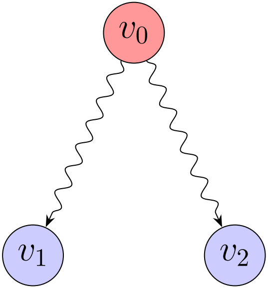
关注某个 n-ary 有序状态空间，三步骤证明“状态对兼容性”



① 任取两个状态节点 v_1 和 v_2

引理 (LCA (Lowest Common Ancestor))

n-ary 有序状态空间中的任意一对状态节点都有**唯一的**最近公共祖先。

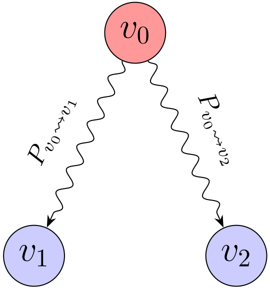


$$v_0 = \mathrm{LCA}(v_1, v_2)$$

② 考虑从 $v_0 = \mathrm{LCA}(v_1, v_2)$ 到 v_1 和 v_2 的两条路径

引理 (Disjoint Paths)

路径 $P_{v_0 \rightsquigarrow v_1}$ 上包含的操作集 $O_{v_0 \rightsquigarrow v_1}$ 与路径 $P_{v_0 \rightsquigarrow v_2}$ 上包含的操作集 $O_{v_0 \rightsquigarrow v_2}$ 不相交。

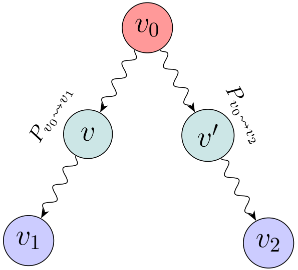


$v_0 = \mathrm{LCA}(v_1, v_2)$

3 考虑两条路径上的状态

引理 (Compatible Paths)

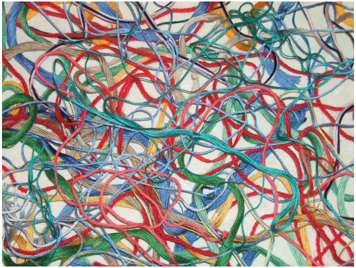
$P_{v_0 \rightsquigarrow v_1}$ 上的任一状态 v 与 $P_{v_0 \rightsquigarrow v_2}$ 上的任一状态 v' 是兼容的。



$$v_0 = \mathrm{LCA}(v_1, v_2)$$

$\therefore$ v_1 和 v_2 是兼容的

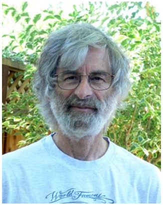
个人体会: 基于 OT 思想的协议晦涩难懂

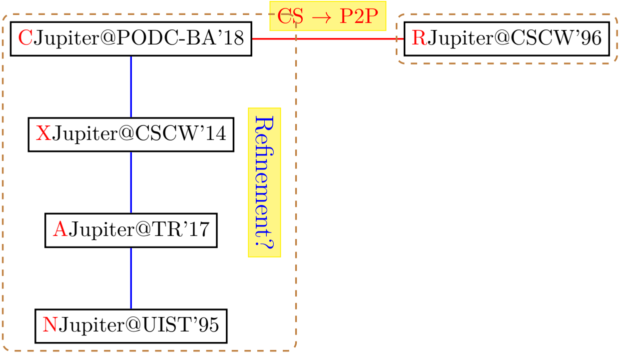


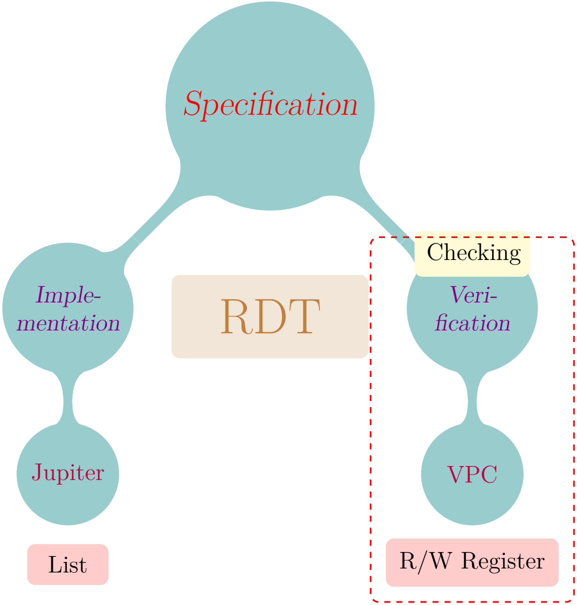
- 协议多种多样
- 经常不加证明
- 证明是错误的
- 勘误也是错的

Model Checking: 使用 TLAPLUS

jupiter-tlaplus@github






Specification
Imple-
mentation
RDT
Checking
Veri-
fication
Jupiter
VPC
List
R/W Register

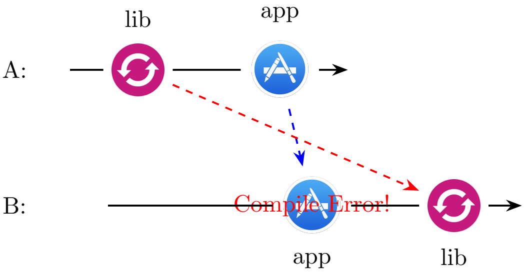
协议验证 (Verification of a Protocol)

执行验证 (Verification of an Execution)

黑盒测试/确认系统是否提供了其所声称的数据一致性

[DeCandia et al., 2007] [Golab, Li, and Shah, 2011]

PRAM: 包含存储系统常提供的最基本的 “会话” (session) 一致性
[Terry et al., 1994] [Brzezinski, Sobaniec, and Wawrzyniak, 2004]



PRAM 保证 “单调写” 性质

定义 (VPC (Verifying PRAM Consistency) 判定问题)

实例: 系统执行 *(execution e)*

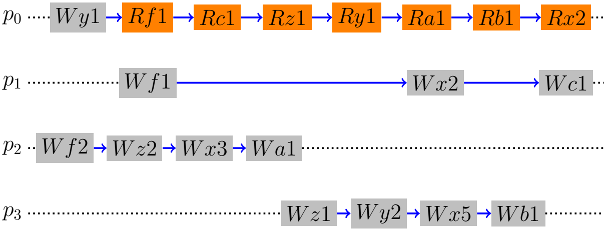
问题: 该执行 e 是否满足 PRAM 一致性模型 $(\mathcal{C})$?

$$e \in \mathcal{C} \Rightarrow \{0, 1\}?$$

定义 (系统执行)

系统执行 $e \triangleq \{h_p \mid h_p : \text{进程 } p \text{ 上的读写操作序列}\}$

规模 n: 系统执行中读写操作的总数

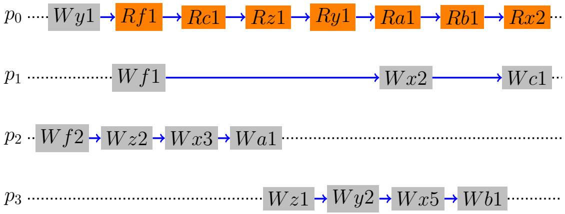


定义 (PRAM 一致性模型)

系统执行 e 满足 PRAM 一致性

$$\Longleftrightarrow$$

$\forall p : p$ 上所有操作与其它进程上所有写操作存在合法调度



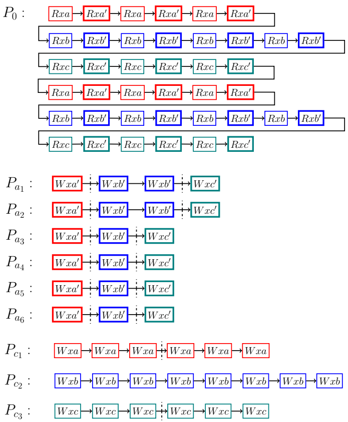
$$p_0 : Wf2\ Wf1\ Wz2\ Wz1\ Wy2\ Wy1\ Rf1\ Wx5\ Wx3\ Wx2\ Wc1\ Rc1\ Rz1\ Ry1\ Wa1\ Ra1\ Wb1\ Rb1\ Rx2$$

VPC 问题的四种变体 (按 “执行” 的类型) 及复杂度 ([∗] : 本文工作)

	(S)ingle variable	*(M)ultiple variables*
write (D)uplicate values	VPC-SD (NP-complete) [∗]	VPC-MD (NP-complete) [∗]
write (U)nique valuewrite (U)nique value	VPC-SU (P) [Golab, Li, and Shah, 2011]	VPC-MU (P) [∗]

Read-mapping [Gibbons and Korach, 1997]: $\forall r,\ \exists! w,\ f(r) = w.$

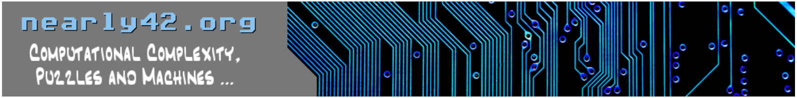
VPC-SD (VPC-MD) 是 NP-complete 问题



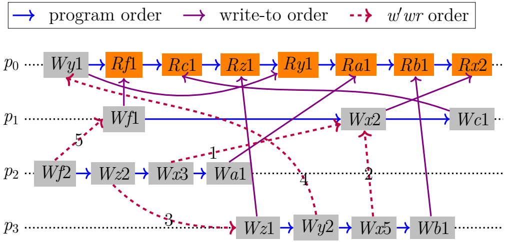
UNARY 3-PARTITION 实例 $A = \{2, 2, 1, 1, 1, 1\}, m = 2, B = 4$ 对应的 VPC-SD 执行

"Basically I'm a programmer :-)

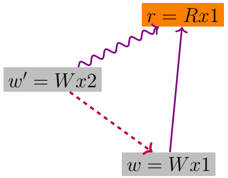
I enjoy reading papers about the complexity of puzzle games, and I'm writing (amateur) proofs on the complexity of a few puzzle games."



VPC-MU 的多项式算法 RW-CLOSURE



RW-CLOSURE 算法示例: 在传递闭包之上迭代应用 w/wr 规则



定理 (RW-Closure 算法正确性)

VPC-MU 实例满足 PRAM 一致性

$$\iff$$

RW-Closure 算法所得图是 DAG 图

证明

"$\Longrightarrow$" 反证法

"$\Longleftarrow$" 对读操作作数学归纳，构造合法调度

RW-Closure 算法复杂度:

$$\underbrace{O(n^2)}_{\#\text{loops}} \cdot \underbrace{O(n^3)}_{\text{transitive closure}} = O(n^5)$$

RW-CLOSURE 算法的缺点:

- 在全图上应用 $w'wr$ 规则
- 应用 $w'wr$ 规则无特定顺序

VPC-MU 的多项式算法 READ-CENTRIC 要点:

- 增量式调度每个读操作
- 在读操作诱导的局部子图上按逆拓扑序应用 $w'wr$ 规则

定理 (Read-Centric 算法正确性)

VPC-MU 实例满足 PRAM 一致性

$$\Longleftrightarrow$$

Read-Centric 算法所得图是 DAG 图

证明

$$\textsc{Read-Centric} \xleftrightarrow{\textit{Reachability}} \textsc{RW-Closure}$$

Read-Centric 算法复杂度:

$$\underbrace{O(n)}_{\#\text{reads}} \cdot \underbrace{O(n \cdot n^2)}_{\textsc{Topo-Schedule}} = O(n^4)$$

VPC 在相关工作中的意义

较早关注（分布式系统领域）“弱一致性模型验证” 问题（2013～）:

- 强一致性: [Gibbons and Korach, 1997] [Cantin, Lipasti, and Smith, 2005] [Golab, Li, and Shah, 2011]
- 弱一致性: [Furbach et al., 2014] [Bouajjani et al., 2017] [Emmi and Enea, 2018]

VSC (Verifying Sequential Consistency) 与 **VL** (Verifying Linearizability) 问题的复杂度 [Gibbons and Korach, 1997]

Variants	**VSC**	**VL**
General	NP-complete	NP-complete
2 Operations/Process	NP-complete	NP-complete
2 Variables	NP-complete	NP-complete
3 Processes	NP-complete	$O(n \log n)$
Read-mapping	NP-complete	$O(n \log n)$
Write-order	NP-complete	$O(n \log n)$
`read&write` only	NP-complete	NP-complete
Conflict-order	$O(n \log n)$	$O(n \log n)$

VMC (Verifying Memory Coherence) 问题的复杂度 [Cantin, Lipasti, and Smith, 2005]

Variants	**Read/Write**	**Read-Modify-Write**
1 Operation/Process	$O(n \lg n)$	$O(n^2)$
2 Operations/Process	?	NP-complete
3+ Operations/Process	NP-complete	NP-complete
Constant k processes	$O(n^k)$	$O(n^k)$
1 Write/Value (Read-mapping)	$O(n)$	$O(n \lg n)$
2 Writes/Value	NP-complete	?
3+ Writes/Value	NP-complete	NP-complete
Write-order	$O(n^2)$	$O(n)$

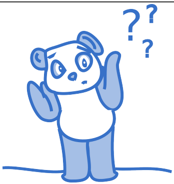
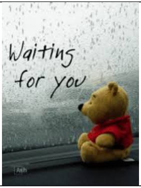
Atomicity [2] 相关一致性模型验证问题复杂度 (假设: 不允许写重复值)

	Safety	**Regularity**	**Atomicity**	**Sequential**
Offline [Anderson et al., 2010]	$O(n^2)$	$O(n^2)$	$O(n^3)$	*not studied*
Online [3] [Golab, Li, and Shah, 2011]	$O(n)$	$O(n)$	$O(n \log n)$	$\mathsf{Poly}(n)$

[2] 也称 Linearizability

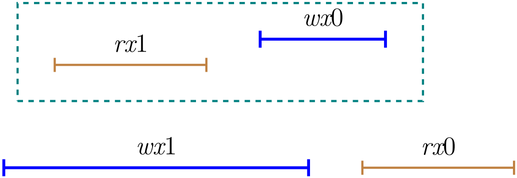
[3] 包含其它假设

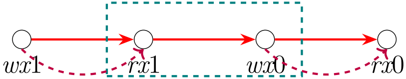
k-AV (k-Atomicity Verification) 问题复杂度

Problems	Variants	Results	Work
1-AV	General	NP-complete	[Gibbons and Korach, 1997]
1-AV	Write unique value	$O(n \log n)$	[Gibbons and Korach, 1997]
2-AV	Write unique value	$O(n \log n)$	[Golab, Hurwitz, and Li, 2013]
k-AV	Write unique value Bounded concurrency	$O(n^2)$	[Golab et al., 2015] [Golab et al., 2018]
k-AV	Write unique value		

$$\boxed{\text{Atomicity} = \text{实时序} + \text{读写语义}}$$

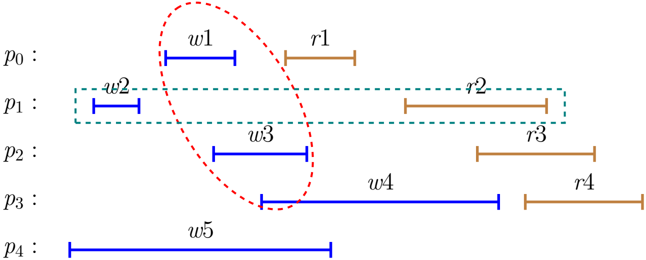
[Lamport, 1986]





k-Atomicity = 实时序 + k-读写语义

[Aiyer, Alvisi, and Bazzi, 2005] [Taubenfeld, 2013]

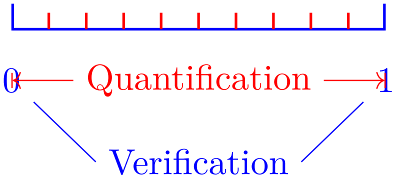


3-Atomicity : $w5$ $w2$ $w1$ $r1$ $w3$ $w4$ $r2$ $r3$ $r4$

定义 (k-AV (k-Atomicity Verification) 判定问题)

实例: 系统执行 e (不允许写重复值)、参数 k

问题: 该执行 e 是否满足 k-Atomicity?

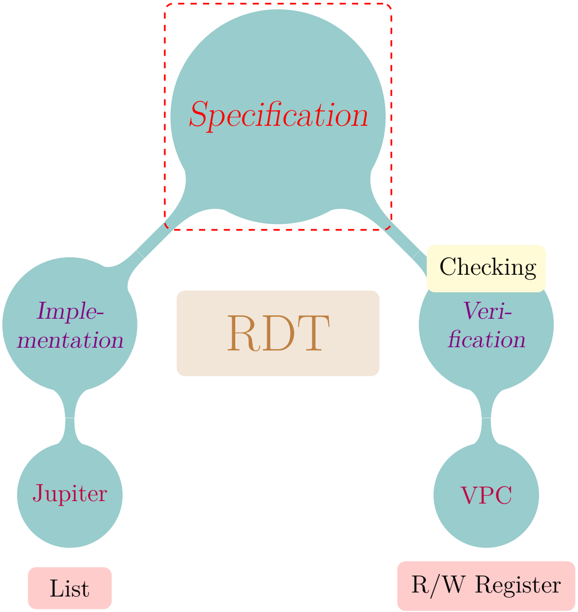


协议量化分析

[Lee and Welch, 2005] [Bailis et al., 2012] [Chatterjee and Golab, 2017]

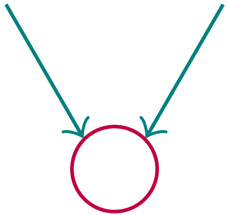
PA2AM: Probabilistically-Atomic 2-Atomicity


One more thing...

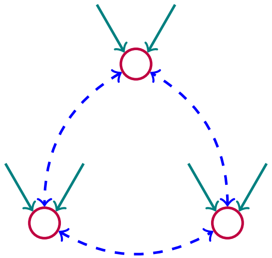

Specification
Imple-
mentation
RDT
Checking
Veri-
fication
Jupiter
VPC
List
R/W Register

规约：数据一致性模型 (Consistency Model)

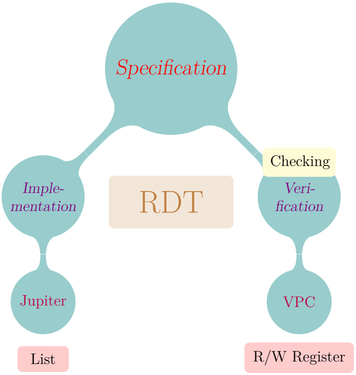
多处理器系统中的并发数据类型



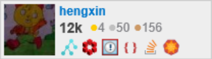
分布式系统中的复制数据类型



PL + DC + FM [Burckhardt et al., 2014]



hfwei@nju.edu.cn



hengxin@homepage hengxin@github hengxin@stackexchange

Aiyer, Amitanand, Lorenzo Alvisi, and Rida A. Bazzi (2005). "On the Availability of Non-strict Quorum Systems". In: *Proceedings of the 19th International Conference on Distributed Computing*. DISC '05. Springer-Verlag, pp. 48–62.

Anderson, Eric et al. (2010). "What Consistency Does Your Key-value Store Actually Provide?" In: *Proceedings of the Sixth International Conference on Hot Topics in System Dependability*. HotDep'10. Vancouver, BC, Canada: USENIX Association, pp. 1–16. URL: http://dl.acm.org/citation.cfm?id=1924908.1924919.

Attiya, Hagit et al. (2016). "Specification and complexity of collaborative text editing". In: *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*. PODC '16. ACM, pp. 259–268.

Bailis, Peter et al. (2012). "Probabilistically Bounded Staleness for Practical Partial Quorums". In: *Proc. VLDB Endow.* 5.8, pp. 776–787.

Bouajjani, Ahmed et al. (2017). "On Verifying Causal Consistency". In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. POPL 2017. Paris, France: ACM, pp. 626–638. ISBN: 978-1-4503-4660-3. DOI: 10.1145/3009837.3009888. URL: http://doi.acm.org/10.1145/3009837.3009888.

Brzezinski, J, C Sobaniec, and D Wawrzyniak (2004). “From session causality to causal consistency”. In: *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pp. 152–158.

Burckhardt, Sebastian (2014). “Principles of Eventual Consistency”. In: *Found. Trends Program. Lang.* 1.1-2, pp. 1–150. ISSN: 2325-1107. DOI: 10.1561/2500000011. URL: http://dx.doi.org/10.1561/2500000011.

Burckhardt, Sebastian et al. (2014). “Replicated Data Types: Specification, Verification, Optimality”. In: *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL ’14. San Diego, California, USA: ACM, pp. 271–284. ISBN: 978-1-4503-2544-8. DOI: 10.1145/2535838.2535848. URL: http://doi.acm.org/10.1145/2535838.2535848.

Cantin, Jason F., Mikko H. Lipasti, and James E. Smith (2005). “The complexity of verifying memory coherence and consistency”. In: *IEEE Transactions on Parallel and Distributed Systems* 16.7, pp. 663–671.

Chatterjee, Shankha and Wojciech Golab (2017). “Brief Announcement: A Probabilistic Performance Model and Tuning Framework for Eventually Consistent Distributed Storage Systems”. In: *Proceedings of the ACM Symposium on Principles of Distributed Computing*. PODC ’17. Washington, DC, USA: ACM, pp. 259–261. ISBN: 978-1-4503-4992-5. DOI: 10.1145/3087801.3087850. URL: http://doi.acm.org/10.1145/3087801.3087850.

DeCandia, Giuseppe et al. (2007). “Dynamo: Amazon’s Highly Available Key-value Store”. In: *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles*. SOSP ’07. ACM, pp. 205–220.

Ellis, C. A. and S. J. Gibbs (1989). “Concurrency Control in Groupware Systems”. In: *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’89. ACM, pp. 399–407.

Emmi, Michael and Constantin Enea (2018). “Monitoring Weak Consistency”. In: *Computer Aided Verification*. Ed. by Hana Chockler and Georg Weissenbacher. Cham: Springer International Publishing, pp. 487–506. ISBN: 978-3-319-96145-3.

Furbach, F. et al. (2014). “Memory Model-Aware Testing - A Unified Complexity Analysis”. In: *2014 14th International Conference on Application of Concurrency to System Design*, pp. 92–101.

Gibbons, Phillip B. and Ephraim Korach (1997). "Testing shared memories". In: *SIAM J. Comput.* 26.4, pp. 1208–1244.

Golab, Wojciech, Jeremy Hurwitz, and Xiaozhou (Steve) Li (2013). "On the k-Atomicity-Verification Problem". In: *Proceedings of the 33rd IEEE International Conference on Distributed Computing Systems*. ICDCS '13. IEEE Computer Society, pp. 591–600.

Golab, Wojciech, Xiaozhou Li, and Mehul A. Shah (2011). "Analyzing Consistency Properties for Fun and Profit". In: *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*. PODC '11. ACM, pp. 197–206.

Golab, Wojciech et al. (2015). "Computing Weak Consistency in Polynomial Time: [Extended Abstract]". In: *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*. PODC '15. ACM, pp. 395–404.

Golab, Wojciech et al. (2018). "Computing k-Atomicity in Polynomial Time". In: 47, pp. 420–455.

Herlihy, Maurice P. and Jeannette M. Wing (1990). "Linearizability: A Correctness Condition for Concurrent Objects". In: *ACM Trans. Program. Lang. Syst.* 12.3, pp. 463–492. ISSN: 0164-0925. DOI: `10.1145/78969.78972`. URL: `http://doi.acm.org/10.1145/78969.78972`.

- Lamport, Leslie (1986). "On interprocess communication (Part II: algorithms)". In: *Distrib. Comput.* 1.2, pp. 86–101.
- Lee, Hyunyoung and Jennifer L. Welch (2005). "Randomized Registers and Iterative Algorithms". In: *Distrib. Comput.* 17.3, pp. 209–221.
- Liskov, Barbara and Stephen Zilles (1974). "Programming with Abstract Data Types". In: *Proceedings of the ACM SIGPLAN Symposium on Very High Level Languages*. Santa Monica, California, USA: ACM, pp. 50–59. DOI: 10.1145/800233.807045. URL: http://doi.acm.org/10.1145/800233.807045.
- Nichols, David A. et al. (1995). "High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System". In: *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*. UIST '95. ACM, pp. 111–120.
- Shapiro, Marc et al. (2011a). *A comprehensive study of Convergent and Commutative Replicated Data Types*. Research Report RR-7506. Inria – Centre Paris-Rocquencourt ; INRIA, p. 50. URL: https://hal.inria.fr/inria-00555588.
- Shapiro, Marc et al. (2011b). "Conflict-free Replicated Data Types". In: *Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems*. SSS'11. Springer-Verlag, pp. 386–400.

Taubenfeld, Gadi (2013). "Weak Read/Write Registers". In: *Distributed Computing and Networking*. Ed. by Davide Frey et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 423–427. ISBN: 978-3-642-35668-1.

Terry, Douglas B. et al. (1994). "Session Guarantees for Weakly Consistent Replicated Data". In: *Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems*. PDIS '94, pp. 140–149.

Viotti, Paolo and Marko Vukolić (2016). "Consistency in Non-Transactional Distributed Storage Systems". In: *ACM Comput. Surv.* 49.1, 19:1–19:34. ISSN: 0360-0300. DOI: 10.1145/2926965. URL: http://doi.acm.org/10.1145/2926965.

Xu, Yi, Chengzheng Sun, and Mo Li (2014). "Achieving Convergence in Operational Transformation: Conditions, Mechanisms and Systems". In: *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work*. CSCW '14. ACM, pp. 505–518.

定义 (最终收敛性 (Eventual Convergence) [Ellis and Gibbs, 1989])

当用户不再提交更新操作时, 所有 *replicas* 上的列表是相同的。

定义 (强最终一致性 (Strong Eventual Consistency) [Shapiro et al., 2011b])

如果两个 *replicas* 处理了同一组更新操作, 则它们的列表是相同的。

对系统的中间状态缺少足够的约束

针对列表的操作转换函数 [Ellis and Gibbs, 1989]

$$OT\Big(\textsc{Ins}(a_1, p_1, pr_1), \textsc{Ins}(a_2, p_2, pr_2)\Big) = \begin{cases} \textsc{Ins}(a_1, p_1, pr_1) & p_1 < p_2 \\ \textsc{Ins}(a_1, p_1 + 1, pr_1) & p_1 > p_2 \\ \text{NOP} & p_1 = p_2 \land a_1 = a_2 \\ \textsc{Ins}(a_1, p_1 + 1, pr_1) & p_1 = p_2 \land a_1 \neq a_2 \land pr_1 > pr_2 \\ \textsc{Ins}(a_1, p_1, pr_1) & p_1 = p_2 \land a_1 \neq a_2 \land pr_1 \leq pr_2 \end{cases}$$

$$OT\Big(\textsc{Ins}(a_1, p_1, pr_1), \textsc{Del}(\_, p_2, pr_2)\Big) = \begin{cases} \textsc{Ins}(a_1, p_1, pr_1) & p_1 \leq p_2 \\ \textsc{Ins}(a_1, p_1 - 1, pr_1) & p_1 > p_2 \end{cases}$$

$$OT\Big(\textsc{Del}(\_, p_1, pr_1), \textsc{Ins}(a_2, p_2, pr_2)\Big) = \begin{cases} \textsc{Del}(\_, p_1, pr_1) & p_1 < p_2 \\ \textsc{Del}(\_, p_1 + 1, pr_1) & p_1 \geq p_2 \end{cases}$$

$$OT\Big(\textsc{Del}(\_, p_1, pr_1), \textsc{Del}(\_, p_2, pr_2)\Big) = \begin{cases} \textsc{Del}(\_, p_1, pr_1) & p_1 < p_2 \\ \textsc{Del}(\_, p_1 - 1, pr_1) & p_1 > p_2 \\ \text{NOP} & p_1 = p_2 \end{cases}$$

定理 (RW-CLOSURE 算法正确性)

VPC-MU 实例满足 PRAM 一致性

$$\Longleftrightarrow$$

RW-CLOSURE 算法所得图是 DAG 图

证明

"$\Longrightarrow$" 反证法

"$\Longleftarrow$" 难点: *DAG* 图蕴含着多个全序
技巧: 对读操作作数学归纳, 构造合法调度

RW-CLOSURE 算法复杂度:

$$\underbrace{O(n^2)}_{\#\text{loops}} \cdot \underbrace{O(n^3)}_{\text{transitive closure}} = O(n^5)$$

READ-CENTRIC 算法复杂度:

$$\underbrace{O(n)}_{\text{iterations}} \cdot \underbrace{O(n \cdot n^2)}_{\text{TOPO-SCHEDULE}} = O(n^4)$$

引理 (TOPO-SCHEDULE 的非迭代性)

设 TOPO-SCHEDULE 正在处理读操作 r,
则局部子图中的每个写操作最多只有一次机会
在满足规则 $w'wr$ 的三元组中扮演 "w'" 角色"。

实验评估

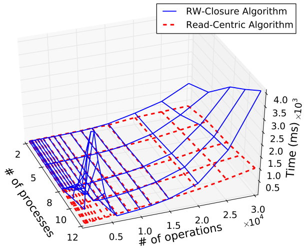
实验目的[1]：

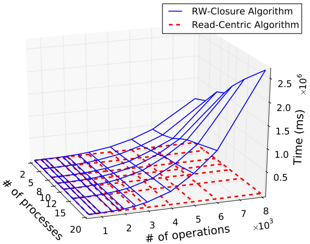
1. 考察 Read-Centric 算法的实际效率 (*vs.* 渐近时间复杂度)
2. 对比 Read-Centric 算法与 RW-Closure 算法的效率

两类负载:

1. 随机生成的系统执行
2. 满足 PRAM 一致性的系统执行 ($\approx$ 最坏情况输入)

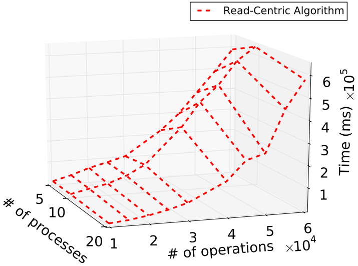
[1]机器配置: Intel Core i7 3.40GHZ, 4GB RAM.





RW-CLOSURE 算法与 READ-CENTRIC 算法在 (左) 随机生成的执行及 (右) 满足 PRAM 一致性的执行上的运行时间。

(右) 20 个进程、8,000 个操作:
READ-CENTRIC 可获得 694 倍加速.



READ-CENTRIC 算法在满足 PRAM 一致性的执行上的运行时间

READ-CENTRIC: 20 个进程、60,000 个操作 $<$ 600s [1]

RW-CLOSURE: 20 个进程、8,000 个操作 $>$ 3,000s

[1] 用于测试, 规模可用

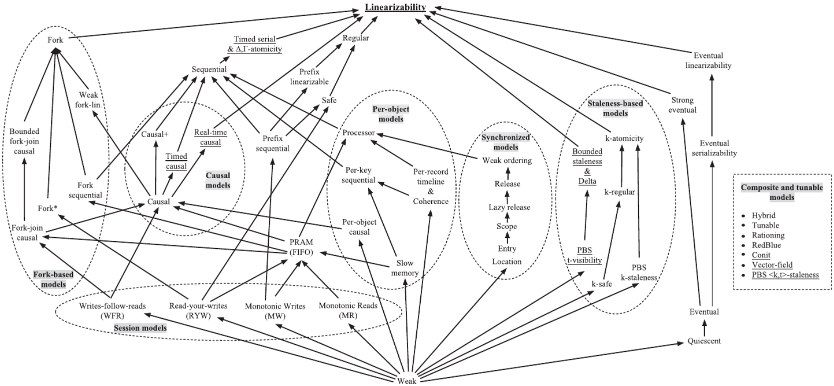
$\mathcal{I}$-Atomicity $=$ i-实时序 $+$ 读写语义

$\mathcal{I}$: Inversions

$$f(\{\text{inversions}\}) \leq i$$

定义框架

(50 种) 一致性模型关系图 [Viotti and Vukolić, 2016] [Burckhardt, 2014]



建立统一的形式化框架