arXiv:2301.02668v1 [cs.DC] 6 Jan 2023

用于天气模拟的经数据同化验证的大规模粒子滤波器框架

Sebastian Friedemann[a]、Kai Keller[b]、Yen-Sen Lu[c]、Bruno Raffin[a,*]、Leonardo Bautista-Gomez[b]

[a]法国格勒诺布尔 *38000*，*Univ. Grenoble Alpes*、*Inria*、*CNRS*、*Grenoble INP*、*LIG*[b]西班牙巴塞罗那 *08034*，巴塞罗那超级计算中心[c]德国于利希 *52428*，*Forschungzentrum Juelich*

摘要

粒子滤波器是一类算法，当模型不符合线性和高斯假设时，可通过统计贝叶斯方法求解逆问题。粒子滤波器应用于数据同化、概率编程、神经网络优化、定位和导航等领域。粒子滤波器通过运行大量模型实例（即所谓的粒子），估计模型状态的概率分布。处理极大量粒子的能力对高维模型至关重要。本文提出了一种新范式，可在超级计算机上运行规模极大的并行模型实例集合。该方法结合了弹性容错运行器/服务器模型，在实现动态负载均衡的同时尽量减少数据移动。每个运行器在本地计算粒子权重，并在权重可用时将其发送给服务器；服务器对权重进行归一化，根据权重重新采样生成新粒子，并动态地将工作重新分配给运行器，以应对负载不均衡。我们的方法依托异步管理的分布式粒子缓存，使粒子能够在后台从一个运行器迁移到另一个运行器，同时粒子传播仍在进行。这还使运行器的数量能够在执行期间发生变化，以应对故障和

*通讯作者电子邮箱：sebastian.friedemann@inria.fr（SebastianFriedemann）、kai.keller@bsc.es（Kai Keller）、ye.lu@fz-juelich.de（Yen-Sen Lu）、bruno.raffin@inria.fr（Bruno Raffin）、leonardo.bautista@bsc.es（Leonardo Bautista-Gomez）

重启，或适应外部决策流程决定的资源可用性变化。我们使用天气研究与预报模式（WRF）对该方法进行了实验，以评估其在概率天气预报中的性能。我们使用多达 2,555 个粒子和 20,442 个计算核心，将云量观测同化到欧洲地区的短期天气预报中。

关键词：资料同化、粒子滤波器、集合运行、韧性、弹性

---

1. 引言

给定一个输出和一个变换函数，求解输入状态就是所谓的**逆问题**。针对这一核心问题，已有各种各样的方法。统计贝叶斯方法尤为突出，因为它们以概率密度函数的形式，为所提出的输入提供不确定性度量。本文考虑**粒子滤波器**这一统计贝叶斯方法，它结合动力系统和观测值的不确定性，以估计系统**状态**。动力系统的若干种实现形式称为**粒子**，它们的内部状态经过不同扰动，并被**传播**至可获得**观测数据**的时刻。随后，根据这些粒子与观测值之间的距离为其**赋予权重**。利用这些权重生成新的粒子样本，使其与观测值更加吻合。只要有观测值可用，这一过程就会持续重复。

粒子滤波器可用于多种用途，例如数据同化（*DA*） [1],概率编程 [2, 3, 4], 神经网络优化 [5], 本地化和导航[6]。与集合卡尔曼滤波（EnKF）等技术相比，粒子滤波器的优势在于能够处理非线性和/或非高斯状态空间模型。但要实现这一优势，就需要运行更多粒子。如果动力系统采用先进的并行高维数值模型求解器（例如地球科学应用中的求解器），则可能需要数千个粒子才能避免欠采样和退化。高维、大规模求解器本身已十分消耗计算资源，而运行数千个实例会使计算量增加数个数量级。粒子滤波器的大规模数据同化，例如用于天气预报等地球科学应用 [7]。如今，达到百亿亿次级的超级计算机拥有支持超大规模

粒子滤波器。然而，要高效利用这些资源，同时节省时间和能源，并非易事。应用程序需要尽量减少对并行文件系统（PFS）的使用，因为它是超级计算机的传统瓶颈；同时，应优先采用原位数据处理和本地数据存储，以减少数据移动，并通过异步处理尽可能重叠各项任务。应用程序还需要具备灵活性，以适应执行期间的变化，因此必须支持韧性、弹性和动态负载均衡。

现有的大规模方法可分为两类：**在线**和**离线**方法。离线方法使用临时文件交换数据。要传播一个粒子，首先启动一个模型实例，从文件加载该粒子，将其传播到指定时间，再将生成的粒子存回文件，最后关闭模型实例。这种方法灵活，且易于支持容错，但性能会受到影响，尤其是在大规模场景下，因为它大量使用文件系统，并且每次传播都要启动一个新的模型实例。在线方法则通过运行一个涵盖完整工作流的大型 MPI 应用程序来绕过文件系统，粒子经由网络通过 MPI 通信分配给不同的模型实例。虽然这种方法节省了 I/O 开销，却牺牲了灵活性。例如，单个粒子传播过程中发生故障，就会导致整个应用程序停止。因此，现有的在线方法通常不支持容错或动态负载均衡，相关工作一节（第 6 节）将对此作详细介绍。

本文提出了一种替代方案，可构建高效且灵活的框架。实现这一目标的关键在于粒子传播的**虚拟化**。我们将一个数值模型求解器实例转化为**运行器**，使其能够依次传播多个粒子，同时将开销和空闲时间降至最低。每个运行器都与节点本地分布式状态缓存相连，从而能够快速加载和存储粒子。缓存会异步持久化到文件系统，以支持检查点和运行器之间的负载均衡。将粒子异步预取到缓存中，可使粒子加载与粒子传播重叠执行。**服务器**负责组织运行器的工作分发，并集中执行粒子滤波器更新和（重新）采样任务。运行器和服务器分别作为独立的可执行程序运行，以支持弹性并简化容错。将这些不同特性与容错协议相结合，可构建一个弹性且可靠的框架，在实现动态负载均衡的同时，最大限度地减少数据移动。粒子虚拟化使资源分配与粒子数相互独立。

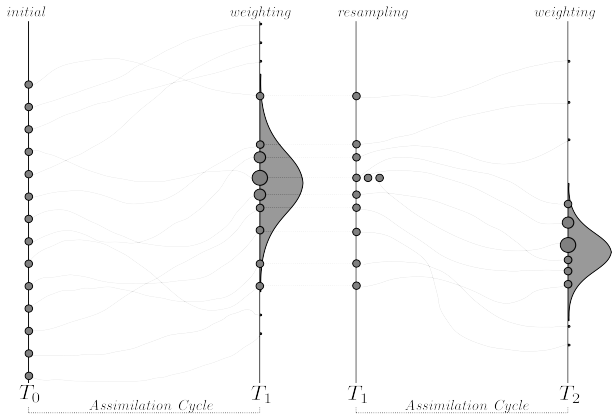


图 1： 初始时，粒子均匀采样。随后将其传播至 T_1，并根据观测数据计算权重。重采样会舍弃一些权重较低的粒子（顶部和底部），而其他权重较高的粒子则会成为多个粒子的父粒子（此处为 3 个）。

运行器数量可在执行期间发生变化，既可以响应故障和重启，也可以适应外部决策过程所决定的资源可用性变化。所提出的架构旨在超大规模环境中运行，并利用当前及未来超级计算机的深层存储层次结构和异构集群设计。

我们通过一个实际应用案例检验所提出的粒子滤波框架，该案例将框架与天气研究与预报（WRF，版本 3.7.1）模型对接 [8]。WRF 是广泛用于业务预报和研究的天气模型。借助我们的粒子滤波器，我们能够在欧洲区域使用 20,442 个计算核心运行 2,555 个粒子的 WRF 模拟，效率达到 87%。

本文其余部分的结构如下：第 2 节回顾粒子滤波器的原理及其相关工作流。第 3 节介绍所提出方法的架构，第 4 节介绍实验，第 5 节则讨论相关内容。本文最后在第 6 节介绍相关工作，并在第 7 节给出结论。

2. 粒子滤波器

本节将简要介绍粒子滤波形式体系，重点介绍我们在所提方案中加以利用的特性。关于更全面的

介绍，请参阅 [1, 9]。设 $\mathcal{M}$ 为一个数值模型，它将粒子 p 从时间 $t-1$ 处的状态 $\mathbf{x}_{p,t-1}$ 传播至时间 t处的状态 $\mathbf{x}_{p,t}$：

$$\mathbf{x}_{p,t} = \mathcal{M}(\mathbf{x}_{p,t-1}) + \boldsymbol{\beta}^t \tag{1}$$

其中，$\boldsymbol{\beta}$ 是表示模型误差的随机强迫。令 $\mathcal{H}$ 为从状态空间到观测空间的投影算子：

$$\mathbf{y} = \mathcal{H}(\mathbf{x}) + \boldsymbol{\epsilon}^t \tag{2}$$

其中，$\boldsymbol{\epsilon}$ 是表示测量误差的随机向量。

自举粒子滤波器形式体系可以通过贝叶斯定理推导：

$$p(\mathbf{x}_t|\mathbf{y}_t) = \frac{p(\mathbf{y}_t|\mathbf{x}_t)\, p(\mathbf{x}_t)}{p(\mathbf{y}_t)} \tag{3}$$

其中，$p(\mathbf{x}_t|\mathbf{y}_t)$是后验概率密度函数（PDF），$p(\mathbf{x}_t)$是先验概率密度函数，$p(\mathbf{y}_t|\mathbf{x}_t)$是在$\mathbf{x}_t$ 代表真实状态的情况下观测到$\mathbf{y}_t$ 的似然，而 $p(\mathbf{y}_t)$是可用的证据。滤波形式体系的目标是推导后验分布 $p(\mathbf{x}_t|\mathbf{y}_t)$，它描述了考虑证据$\mathbf{y}_t$后状态$\mathbf{x}_t$ 的概率密度函数。

在自举粒子滤波器中，通过对一组 P 粒子$\mathbf{x}_{p,t}$ 进行采样来估计先验 $p(\mathbf{x}_t)$，这些粒子代表不同的模型状态

$$p(\mathbf{x}_t) = \frac{1}{P}\sum_{p=0}^{P-1} \delta(\mathbf{x}_t - \mathbf{x}_{p,t}), \tag{4}$$

假定似然 $p(\mathbf{y}_t|\mathbf{x}_t)$已知，并在传感器校准时进行估计。它根据 $\boldsymbol{\epsilon}$ 的概率密度函数推导得出，该函数应用于状态与观测$\mathbf{y}_t - \mathcal{H}(\mathbf{x}_t)$之间的距离：

$$p(\mathbf{y}_t|\mathbf{x}_t) = p_\epsilon(\mathbf{y}_t - \mathcal{H}(\mathbf{x}_t)) \tag{5}$$

证据 $p(\mathbf{y}_t)$ 可通过以下方式计算：

$$p(\mathbf{y}_t) = \int p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t)\, d\mathbf{x}_t \tag{6}$$

$$= \sum_{p=0}^{P-1} \frac{1}{P}\, p(\mathbf{y}_t|\mathbf{x}_{p,t}) \tag{7}$$

$$\tag{8}$$

将所有内容合并，并将表达式代入贝叶斯定理（公式 3），即可得到后验分布 [1] 的表达式：

$$p(\mathbf{x}_t|\mathbf{y}_t) \approx \sum_{p=0}^{P-1} \hat{w}_{p,t}\, \delta(\mathbf{x}_t - \mathbf{x}_{p,t}) \tag{9}$$

其中，$\hat{w}_{p,\ t}$ 为粒子权重的归一化值：

$$\hat{w}_{p,t} = \frac{p(\mathbf{y}_t|\mathbf{x}_{p,t})}{\sum_{q=0}^{P-1}\ p(\mathbf{y}_t|\mathbf{x}_{q,t})} = \frac{w_{p,t}}{\sum_{q=0}^{P-1}\ w_{q,t}} \tag{10}$$

并且，$w_{p,\ t}$ 为粒子权重的未归一化值：

$$w_{p,t} = p(\mathbf{y}_t|\mathbf{x}_{p,t})\, w_{p,t-1} \tag{11}$$

注意，初始权重均设为 $w_{p,\ 0} = 1/P$。

尤其是在高维模型中，粒子滤波器往往会受到权重退化的影响，即一个归一化权重接近于 1，而其他权重都接近于 0。对抗集群退化的一种经典方法是序贯重要性重采样（SIR） [10, 11]。SIR 的流程是在传播步骤结束时，从后验分布（公式 9）中重采样粒子； P 粒子根据各自的概率 $w_{p,\ t}$ 从现有粒子中随机抽取并重采样。权重较低的粒子会被丢弃，而权重较高的粒子可能成为多次粒子传播的起点（图 1）。更准确地说，重采样会生成多重集 P，该多重集由有序对 $(Q,\ \boldsymbol{\alpha})$ 定义。其中， Q 是 P 中唯一粒子 q 的集合，而 α_q 是 q 在P中出现的次数。粒子 q 在下文中称为**父粒子**。

重新采样的粒子会再次被赋予相同的权重 $w_{p,\ t} = 1/P$。如果模型本身不包含随机成分，那么从同一父粒子出发的粒子可能需要接受随机扰动。否则，这些粒子的轨迹将完全相同。

SIR 和重采样算法有多种变体，例如残差重采样 [12]。有些算法会在每个传播阶段后执行重采样步骤，而另一些则根据权重方差等标准决定是否重采样。本文采用在每个传播阶段后进行重采样的 SIR。

框 1

(a) 粒子 p 的传播仅取决于其关联状态 $\mathbf{x}_{p,\ t}$，且可独立于其他粒子进行。(b) 权重 $w_{p,\ t}$ 仅取决于其关联粒子 p 和观测向量 y_t，可独立于其他权重和粒子进行计算。(c) 滤波器更新仅取决于权重 $w_{p,\ t}$，而与粒子及其关联状态无关。(d) 与粒子 p 关联的状态 $\mathbf{x}_{p,\ t}$ 在滤波器更新期间保持不变。

框 1 列出了构成我们实现基础的粒子滤波器属性。

我们利用属性 (d)：与集合卡尔曼滤波等其他数据同化技术不同，在滤波器更新期间，粒子状态保持不变。与观测值偏离过大的粒子（权重较低）会被丢弃，样本集则会收敛到表现最佳的粒子（权重较高）附近。然而，这些粒子对应的状态不会改变。属性 (a) 可直接由公式 11 得出。属性 (b) 则源于将权重计算与滤波器更新解耦（去中心化）。更新本身仅包括权重归一化和粒子重采样。最后，属性 (c) 是自举粒子滤波器的固有属性，因为粒子只会被移除或选中，不会被改变。在接下来的章节中，我们将说明如何利用这些属性，提高粒子滤波器实现的效率和韧性。

3. 架构

本节将详细介绍所提出的架构，该架构用于通过并行数值模型运行大量粒子。第 2 节所述的算法由两个主要步骤依次组成：

1. 第一步是计算密集型的大规模并行处理，在此步骤中，粒子可以并发处理，以实现：

   (a) 传播每个状态：$\mathbf{x}_{p,t} = \mathcal{M}(\mathbf{x}_{p,t-1})$,

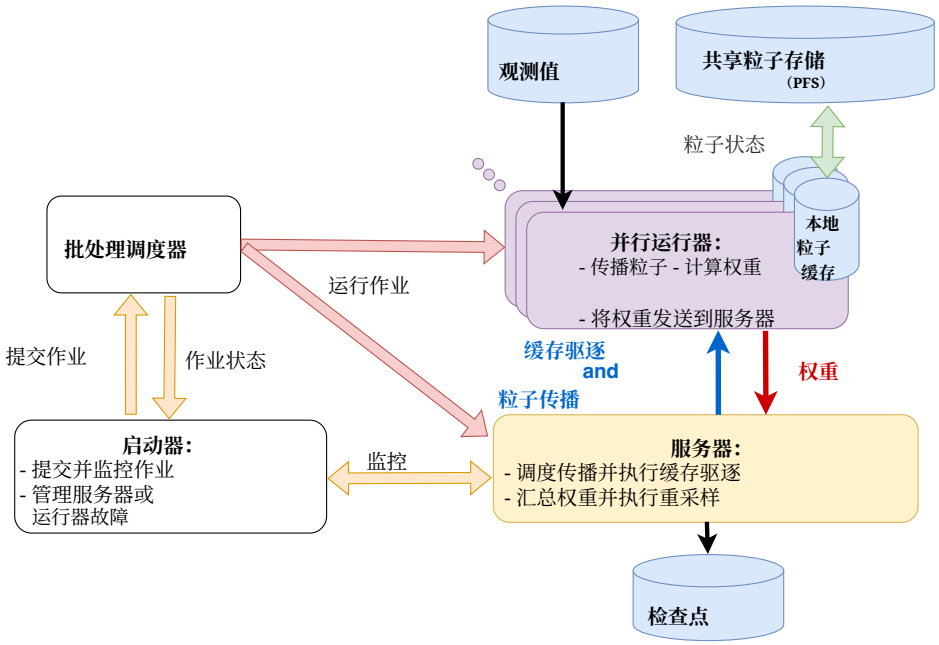


图 2: 架构概览。

(b) 根据每个状态和观测数据计算各自的未归一化权重:

$$w_{p,t} = p_\epsilon(\mathbf{y}_t - \mathcal{H}(\mathbf{x}_{p,t})) \tag{12}$$

2. 第二个轻量级步骤需要收集所有未归一化权重$w_{p,\ t}$，通常每个权重占一个双精度数，用于归一化和重采样。

我们将第一步的工作交给运行器，将第二步交给**服务器**。**运行器**旨在依次传播多个粒子，同时将开销和空闲时间降至最低（图 2）。每个运行器都与一个节点本地的分布式缓存相连，从而能够快速加载和存储粒子。缓存会异步持久化到全局文件系统，以便进行检查点和动态负载均衡（即确保粒子在全局范围内可用）。由于重采样可能会丢弃某些粒子，或复制其他源自同一父粒子的粒子（必要时进行局部扰动），因此需要动态地将状态重新分配给运行器，使它们的工作负载保持均衡。服务器负责将粒子传播任务动态分配给运行器。运行器使用分布式缓存从文件系统加载

缺失的状态。这种设计确保服务器与运行器之间的通信量较低，并减少状态迁移。运行器和服务器作为独立的可执行程序运行，因此运行器数量可以动态变化。这是实现容错和弹性的关键特性。弹性（有时也称为可塑性）是指在资源可用性不断变化的情况下运行的能力；在此处，变化的是运行器数量。

下面将详细介绍这一设计：运行器（第 3.1 节）、服务器（第 3.2 节）、分布式缓存（第 3.3 节）、这些组件之间的工作流（第 3.4 节）、粒子传播调度（第 3.5 节）、作业监控（第 3.6 节）以及容错协议（第 3.7 节），最后介绍更多实现细节（第 3.8 节）。

3.1. 运行器

运行器由模拟代码构成，这些代码通常是高级并行代码，甚至是多个并行代码的耦合体；启动时需要较长时间来加载并构建各种内部数据结构。为避免每次进行粒子传播都要承担重启开销，我们为模拟代码添加了存储和加载粒子状态的机制。这就是**粒子虚拟化**的基础：运行器可以加载一个粒子、对其进行传播、存储结果，并根据需要重复这一过程。运行器与分布式缓存相关联，以加快状态加载和存储，详见第 3.3 节。运行器还会计算相应的权重$w_{p,\ t}$。因此，每个运行器还需要在每个周期加载一次观测值$\mathbf{y}_t$。请注意，观测值的大小通常远小于状态的大小$\mathbf{x}_{p,\ t}$。

3.2. 服务器

服务器负责多项任务。首先，它负责将粒子传播调度给运行器（第3.5节）。其次，它收集运行器发送的权重，并在每个同化周期结束时执行重采样。第三，它控制分布式粒子缓存的内容（第3.3节）。为收集权重$w_{p,\ t}$，每次传播后，运行器都会向服务器发送消息。如果当前周期中仍有粒子需要传播，服务器就会通过消息回复一个唯一标识某个粒子的 ID（下文称为粒子 ID），以供下一次传播使用。否则，服务器会执行重采样，并通过将采样粒子调度给运行器来启动新周期。交换的数据很少，

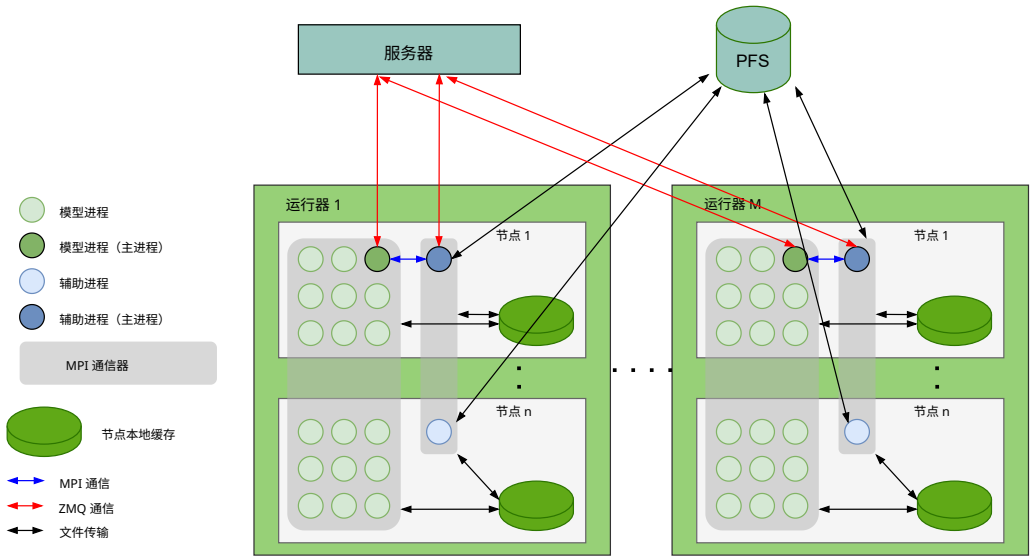


图 3： 运行器内部架构及其与服务器和全局存储（PFS）的交互。与服务器的通信结合了消息传递接口（MPI）和零消息队列（ZMQ）数据交换。

运行器与服务器之间交换的数据很少。运行器发送粒子 ID（一个整数）及其对应的权重（一个浮点数），服务器则回复下一个要传播的粒子 ID。

3.3. 分布式粒子缓存

为了让同一粒子能够在不同运行器上进行多次传播，必须使其全局可用。一种直接的方法是将粒子存储在全局存储中。然而，在超级计算机上，由于工作负载较高且带宽有限，全局存储的吞吐量波动很大。另一方面，节点本地存储仅供运行在相应计算节点上的进程使用，而且带宽可以叠加。将粒子存储在本地，可实现可扩展的 I/O 性能，且性能随计算节点数量线性增长。

为了利用节点本地存储，同时仍使粒子全局可用，运行器依赖于**分布式粒子缓存**。除**模型进程**外，每个运行器还会运行**辅助进程**（每个计算节点一个）；两组进程各自关联一个 MPI 通信器（图 3）。模型进程传播粒子，并将相关状态存储在分配给该运行器的计算节点本地（使用 RAM 磁盘，或在可用时使用其他节点本地存储）。随后，辅助进程会暂存本地的状态

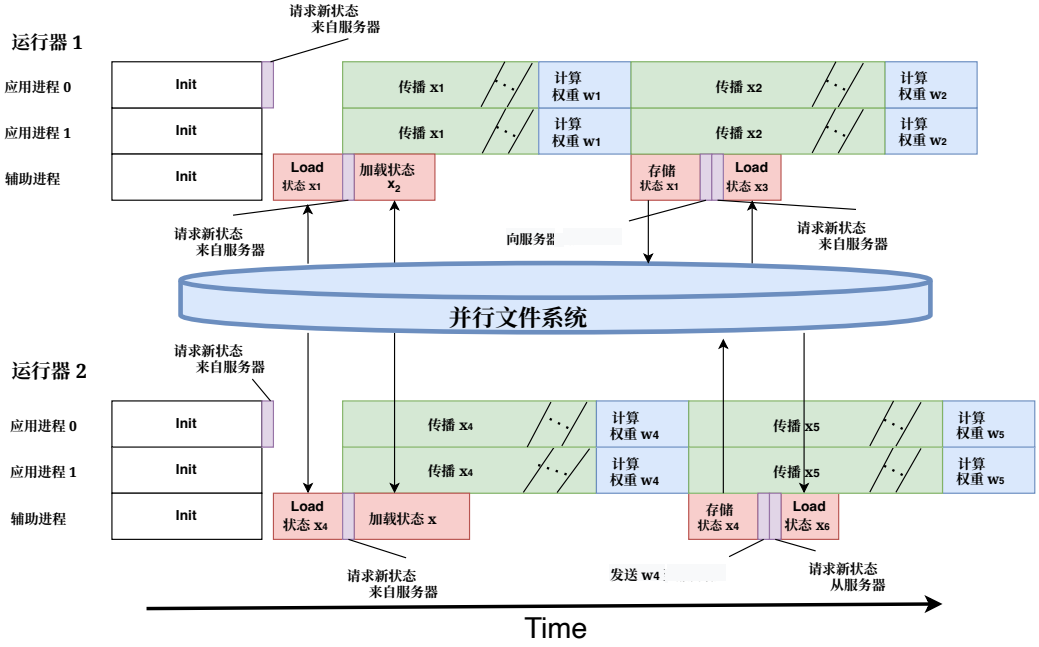


图 4： 示意性甘特图，展示了两个运行器的活动（先进行初始化，随后进行两个同化周期）。重点关注辅助进程的异步加载和存储如何使并行文件系统访问得以隐藏。为简化起见，此处未使用缓存。

异步写入全局存储，从而使相关 I/O 开销能够重叠（图 4）。还请注意，将粒子持久化到全局存储中，可作为容错协议使用的粒子检查点（第 3.7 节）。

我们允许在每个运行器缓存中保留一定数量的粒子，以利用框 1 中的性质 (d)：重采样不会改变粒子状态。因此，在缓存中保留传播后的粒子，可以提高未来传播时（即下一个周期中）在本地找到粒子的概率。如果本地缓存中有可用粒子，运行器便可直接传播该粒子，而无需从全局存储加载。为进一步减少缓存加载，运行器采用了优化的缓存淘汰策略。当一个周期内传播的粒子累计大小超过缓存容量时，淘汰策略尤为重要。由于运行器不了解粒子滤波的状态（传播、重采样），缓存淘汰由服务器控制，并由服务器指示运行器执行。

如第 3.4 节所述，每当模型进程成功完成传播并将粒子存入缓存后，辅助进程便会在后台将其复制到全局存储。因此，所有已传播的粒子状态都可以被选中淘汰，因为它们已安全存储在全局存储中。需要淘汰时，服务器会按以下顺序从缓存中选择粒子：

1. 重采样时从上一周期中丢弃的粒子；2. 当前周期中所有相关传播均已完成且已收到所有权重的父粒子；3. 当前周期传播的权重最低的粒子；4. 随机选取的粒子。

对于情况 1 和 2，可以安全地从缓存中移除这些粒子状态，因为这些粒子后续不再需要用于传播。对于情况 3，我们选择权重最低的粒子状态，因为在重采样过程中，它在后续周期中被选中的概率最低。

3.4. 运行器/服务器工作流

运行器作业启动后，会动态连接到服务器，并请求一个要从该服务器传播的粒子。服务器按照第3.5节所述的调度策略选择粒子，

。模型检查粒子的位置。如果粒子已位于本地缓存中，则开始传播。否则，模型进程会请求辅助进程将状态加载到缓存中。模型进程会一直阻塞，直到辅助进程将粒子加载到缓存中，随后开始传播。

粒子传播完成后，模型会计算相应的权重 w_p ，并将粒子存入缓存。此外，权重和粒子 ID 会发送给辅助进程，同时请求一个新粒子进行传播。辅助进程收到模型进程发送的权重后，会将粒子从缓存暂存到全局存储，随后将权重和粒子 ID 发送给服务器。这样的顺序可确保服务器只有在相应粒子完成传播并成功存入全局存储后，才会收到权重。

辅助进程还会与传播并行地预取粒子（图 4）。这样做是为了避免模型进程在等待从全局存储加载粒子（缓存未命中）时被阻塞。每当辅助进程向服务器发送一个权重时，也会请求下下个要传播的粒子 ID。该粒子会被预取到缓存中，以便在随后进行传播时可在本地使用。每个传播周期结束时都会暂停预取，因为只有在服务器完成所有粒子的重采样后，才能确定下一个周期的传播工作。请注意，如果预取的粒子在此期间被分配给另一个运行器，辅助进程可能需要取消预取；此时，模型进程会在等待下一个待传播粒子时处于空闲状态。服务器在做出此类决策以更好地平衡工作负载时，也会确保运行器之间的一致性。事实证明，全局预取在将粒子状态加载与传播重叠方面非常高效（第 4.2 节）。

3.5. 粒子传播调度

本节介绍服务器上实现的调度算法，该算法将粒子传播任务分配给各个运行器。该算法旨在确保运行器之间负载均衡，并尽量减少全局粒子加载，即将粒子从全局存储传输到本地存储。

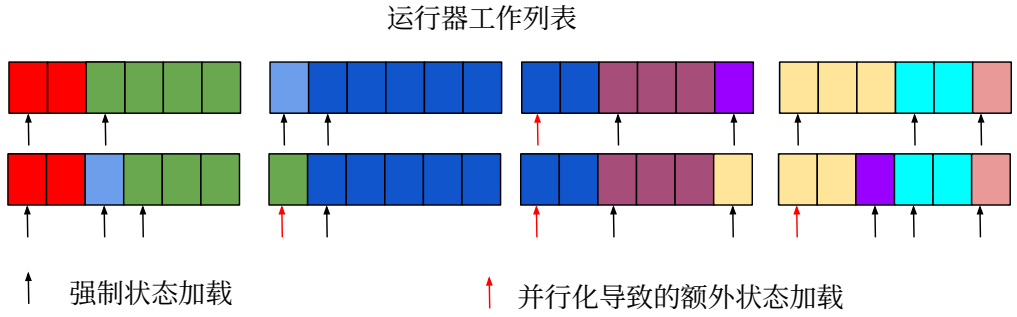


图 5： 在 4 个运行器上调度 24 个持续时间相同的传播任务的两种可能方案。由同一父粒子状态传播而来的所有粒子颜色相同（此处共有 9 个父粒子）。上方的调度方案是最优方案，共需 9 次强制加载（每个父粒子一次），另有一次针对无法由单个运行器容纳的深蓝色父粒子的加载。下方的调度方案多出 2 次状态加载，是我们的在线调度算法可能生成的一种方案。该方案虽非最优，但仍低于一般的 $Q+R-1$ 界限，因为算法可确保“颜色切分”不超过 $R-1$ 次，并避免同一运行器针对给定父粒子状态重复加载。

3.5.1. 静态调度

设 R 为运行器数量。设 P 为需要传播的粒子数。重采样可能会导致某些父粒子被抽取多次进行传播。设 Q 为父粒子 q的数量， α_q 为粒子 q 需要传播的次数。需要传播的粒子总数为：

$$P = \sum_{0 \le q < Q} \alpha_q \tag{13}$$

为了评估调度算法的性能，我们首先推导静态情况下粒子加载最小数量 c^* 的下界和上界，其中：(i) 运行器不缓存状态，(ii) 运行器数量保持不变，且 (iii) 所有粒子传播耗时相同。在这些条件下，每个运行器传播 $\frac{P}{R}$ 个粒子。由于没有本地缓存，每个父粒子 q 都至少需要加载一次。因此，强制粒子加载数量为 Q。如果对于所有 q， $\alpha_q = 1$ ，也就是说，每个粒子只抽取一次，那么 $c^* = P$。否则，在 R 个运行器上并行传播可能需要多个运行器加载某些粒子，因此会产生超出强制加载数量的额外粒子加载。事实上，每个粒子 q 至少需要提供给 s_q 个运行器，

其中

$$s_q = \left\lceil \frac{\alpha_q}{\frac{P}{R}} \right\rceil. \tag{14}$$

分配给 R 个运行器后， P 个粒子的列表会被切分 $R-1$ 次。因此，额外的粒子加载最多为 $R-1$。图 5 展示了这一点。如果所有粒子都从同一个父粒子（$Q=1$）传播而来，就会达到这个上限。因此，粒子加载的最小数量可紧密地界定为

$$Q \le c^* \le Q + R - 1. \tag{15}$$

我们可以采用满足该上限的静态调度：每个运行器分配 $\frac{P}{R}$ 个粒子，其中每个父粒子 q 至多分配给 s_q 个运行器，并规定运行器必须完成当前粒子相关的所有传播，才能切换到下一个粒子。

3.5.2. 动态列表调度

不过，我们的目标是处理更一般的情况。现在我们放宽最初的假设，考虑运行器数量可能变化，而且粒子的传播时间可能有显著差异，并且事先未知（但仍不考虑缓存）。在此情境下，我们采用经典的动态列表调度算法：运行器空闲时，会向服务器请求工作，服务器则返回一个待传播的粒子 ID。在一般情况下，列表调度算法保证其耗时至多为最优调度的两倍；而最优调度需要预先知道粒子传播时间 [13, 14]。我们不再盲目选择下一个要传播的粒子，而是调整静态调度策略来进行粒子选择，目标是限制粒子加载数量。调度基于分裂因子 s_q （公式 14）。不过，我们会调整静态调度以适用于动态情况：每次都根据 α_q、 P 和 R 的更新值重新计算 s_q 。要在服务器上实现该算法，我们需要记录当前正在传播粒子 q 的运行器 R_q 的数量，以及粒子 q 剩余传播次数 α_q 。设 r 为请求传播粒子的运行器，粒子分配算法如下：

1：如果 $\alpha_q > 0$ 粒子 q 上次由 r传播，则递减 α_q 并再次分配 q。如果 $\alpha_q = 0$ ，则继续执行 (2)；

2：选择另一个粒子 q' ，其 $\alpha_{q'} > 0$；

3：计算分裂因子 $s_{q'}$。如果 $R_{q'} < s_{q'}$，则赋值 q'，递增 $R_{q'}$，并递减 $\alpha_{q'}$。如果 $R_{q'} = s_{q'}$，则继续执行（2）。

请注意，当服务器检测到某个运行器丢失时，需要更新簿记信息，以便重新纳入该运行器此前正在传播的粒子。

在传播时间相同且运行器数量固定的情况下，该算法产生的分布与静态调度相同，并满足公式15的上界。

3.5.3. 缓存感知调度

现在，我们去掉最后一个假设，提出一种考虑粒子缓存的调度策略。这是一种基于前述策略构建的启发式方法，并通过多项实验进行了验证。粒子选择策略如下：

1. 选择一个已加载到运行器缓存中的父粒子 p_i（缓存命中）；

2. 选择一个不在任何运行器缓存中的父粒子 p_i（缓存未命中）；

3. 选择一个满足分裂因子准则的粒子 p_i（缓存未命中）；

4. 选择分裂因子最大的父粒子 p_i s_i（即使这会违反分裂因子准则）（缓存未命中）。

前三项符合第 3.5.2 节提出的调度策略。第一项优先保留缓存中已有的粒子，只有在需要为加载粒子腾出空间时才将其逐出。接下来的两项遵循第 3.5.2 节的策略，优先处理此前尚未传播的粒子。这样做的理由是尽可能早地开始处理新的父粒子，并在它们进入缓存后，按需尽可能多次地传播，以减少拆分的需求。最后一项偏离了第 3.5.2 节的策略，但实验表明，加入这一项十分有效。这种情况会出现在周期即将结束时。实践证明，即使需要额外加载，保持运行器繁忙也是一种有效的策略，有助于改善负载均衡，从而缩短完成时间。

3.6. 作业提交与监控

工作流由**启动器**控制。启动器是用户配置并启动应用程序的入口。启动器首先启动，并负责启动和监控运行器及服务器实例；它们均在独立的可执行程序/作业中运行。根据容错协议（第 3.7 节）的要求，或出于弹性考虑，启动器还负责终止并重启运行器或服务器。

启动器与机器的作业调度器（例如 Slurm 或 OAR）紧密交互。启动器可配置为向批处理调度器提交作业，每个运行器和服务器各提交一个作业。这种策略为批处理调度器优化机器资源分配提供了最大灵活性，但执行进度会变得非常依赖机器的可用性。用户可能需要更好地控制并发运行的运行器数量。在这种情况下，可以将启动器设置为向批处理调度器申请一个或多个大型资源分配，并在每个资源分配中安排多个运行器实例。为支持此功能，启动器依赖 Slurm salloc/srun [15], 或 OAR 容器[16]的组合。对于更灵活的方案，我们计划支持基于工作流试运行的调度器，例如 Radical-Pilot [17] 或 QCG-PilotJob [18]。

3.7. 容错

容错依赖于快速识别运行器和服务器实例的故障。运行器故障通过两种不同的方式检测。启动器会识别运行器崩溃，并通过集群调度器监控其执行情况。服务器则通过粒子传播的超时来识别无响应的运行器。如果传播超过超时时间，服务器会请求启动器终止相应的运行器。在这两种情况下，启动器最终都会启动一个新的运行器实例。新运行器会连接到服务器并请求工作。如果运行器发生故障，服务器会取消正在进行的传播；传播耗时以及识别运行器故障所花费的时间都会损失。

服务器故障的检测方式类似：服务器崩溃时可直接检测到，或者在服务器超时未响应时检测到。超时通过启动器与服务器之间定期交换信号（心跳信号）来判定。如果服务器发生故障，启动器会终止所有运行器实例，并重启整个框架。服务器会频繁将传播状态存储到检查点中；发生故障时，框架可以恢复到最近一次成功传播的位置。

最后，服务器通过监控启动器与服务器之间的心跳连接来检测启动器故障。如果未收到心跳信号，服务器会将当前粒子状态保存到检查点，然后关闭。同时，运行器会检测到服务器崩溃并关闭，同样使用超时机制。运行器或服务器发生故障时，系统会自动重启；但如果启动器发生故障，则需要手动重启框架。

3.8. 实现细节

启动器和服务器使用 Python 开发。运行器依赖于经过框架 API 插桩的模拟代码，支持 C/C++、Fortran 和 Python。该实现复用了为 EnKF DA [19] 开发的框架中的一些软件组件，例如启动器。分布式缓存的实现依赖于容错接口（FTI） [20]。FTI 是一个多级检查点重启库，支持将检查点异步写入全局存储。对 FTI 进行的主要修改之一与其事件循环有关。事件通过应用程序与 FTI 进程之间的 MPI 通信触发，并由标签标识。为扩展这一机制，我们启用了回调函数注册功能。该回调函数在事件循环中调用，并可使用唯一标签触发用户定义事件。借此，应用程序的检查点便可利用 FTI 提供的所有可靠性级别，同时还能在专用的 FTI 进程上实现缓存管理。

辅助进程与模型进程之间的通信依赖于异步 MPI 消息。为提高效率，与服务器的通信分为两个步骤实现。只有应用程序（即模型）通信器中的秩 0（主进程）和辅助进程通信器中的秩 0（主进程）会与服务器通信。由于需要动态连接，每个主进程都通过零消息队列库使用套接字连接到服务器。辅助进程或模型进程之间需要传播的信息，则依赖于相应通信器中的 MPI 集合通信（图 3）。

框架代码可在 `https://gitlab.inria.fr/melissa/melissa-da` 获取。

4. 实验

*4.1. WRF*应用案例

实验基于成熟的数值天气预报（NWP）系统——天气研究与预报模式（WRF）（V3.7.1）[8]。WRF 的核心是求解完全可压缩的非静力方程，并包含完整的科里奥利项和曲率项，以及一系列物理选项。模拟区域覆盖欧洲大部分地区（见图6），由 220 × 220 个网格单元组成，水平分辨率为 15 km，垂直方向有 49 层，厚度不均。我们对任意日期（2018-10-12）进行提前一天的天气预报（24 小时初始化时间加 48 小时预报时间），时间步为 24 秒或 100 秒。模式采用 WSM6微物理方案、MYNN2边界层物理方案、Grell-3积云参数化方案、Eta Monin-Obukhov相似理论地表层过程和 RUC陆面模式。此外，还启用了非静力选项，以便更详细地模拟云和降水。输入、初始和边界条件基于欧洲中期天气预报中心（ECMWF）重新分析的 ERA5数据集，该数据集每 3 小时更新一次。使用云量（CFRACT）进行资料同化。通过与欧洲气象卫星组织（EUMETSAT）二级卫星数据中的观测云掩膜进行比较，确定粒子权重。模拟得到的云量被转换为云掩膜，观测云掩膜数据则被上采样至模式网格单元的尺寸，以供后续应用。数据按小时提供，因此一个同化周期包含 150（36）个模式时间步（$150 \times 24\,\mathrm{s} \mathrel{\widehat{=}} 1\,\mathrm{h}$ 或 $36 \times 100\mathrm{s} \mathrel{\widehat{=}} 1\,\mathrm{h}$）。

本文所述实验采用了我们提出的粒子滤波（PF）实现方案，在欧洲区域使用的样本量最高达2,555个粒子。在此情况下，我们使用了Jean-Zay超级计算机上512个计算节点的20,442个计算核心。这些计算节点配备了2个 Intel Cascade Lake 6,248 处理器，每个节点总计拥有40个核心，主频为2.5 GHz，并配有192 GiB随机存取存储器。计算节点之间通过 Intel Omni-Path（100 GB/s）连接，全局文件系统采用带有固态硬盘（GridScaler GS18K SSD）的 IBM Spectrum Scale（前身为 GPFS）并行文件系统。所有实验的节点本地缓存均存储在 RAM 磁盘上。表1列出了主要实验的参数。

欧洲区域中与单个粒子相关的气象状态包含 2.5 GiB 数据。因此，2,555 个粒子在

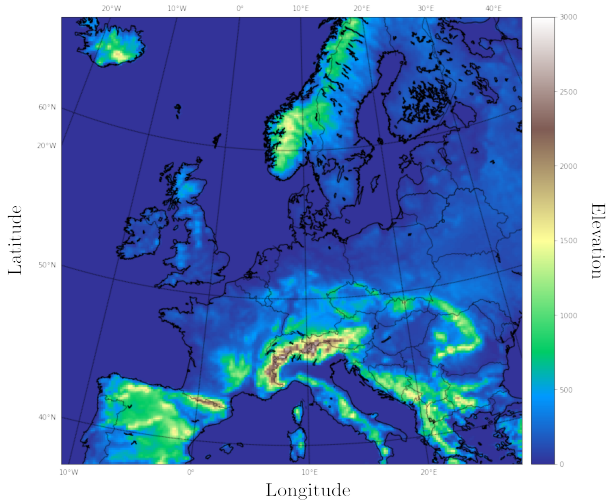


图6：欧洲目标模拟区域的地形。

完整的 48小时模拟周期（48 个时间步）所产生的数据总量约为 300 TiB。本文所进行的实验（包括小规模测试阶段实验）共耗费约 900,000 CPU小时，分配在 JUWELS、Jean-Zay 和 Marenostrum 超级计算机上。

4.2. 运行器活动

本地缓存结合缓存感知调度，可大幅减少数据在全局文件系统层与本地文件系统层之间的传输。缓存命中率，即缓存中找到的粒子数与粒子加载总数之比，取决于缓存大小和每个运行器的粒子数之比。图7展示了不同缓存大小下缓存未命中次数的变化。我们的实验表明，对于128个粒子、32个运行器和大小为9个粒子的缓存，缓存命中率达到88%。这意味着全局存储到本地存储之间的传输量减少了88%。图8直观展示了缓存命中与未命中的模式。初始阶段主要用于启动运行器，所有粒子都从全局存储中获取（缓存预热）。但从下一个同化周期开始，全局存储到本地存储之间的低传输率便逐渐形成。

运行器旨在将对并行文件系统的输入/输出操作与其他任务分开：模型进程仅执行本地输入/输出操作。我们在实验中观察到，这带来了较高的计算效率。与计算任务相比，本地输入/输出访问可以忽略不计（< 0.1 s

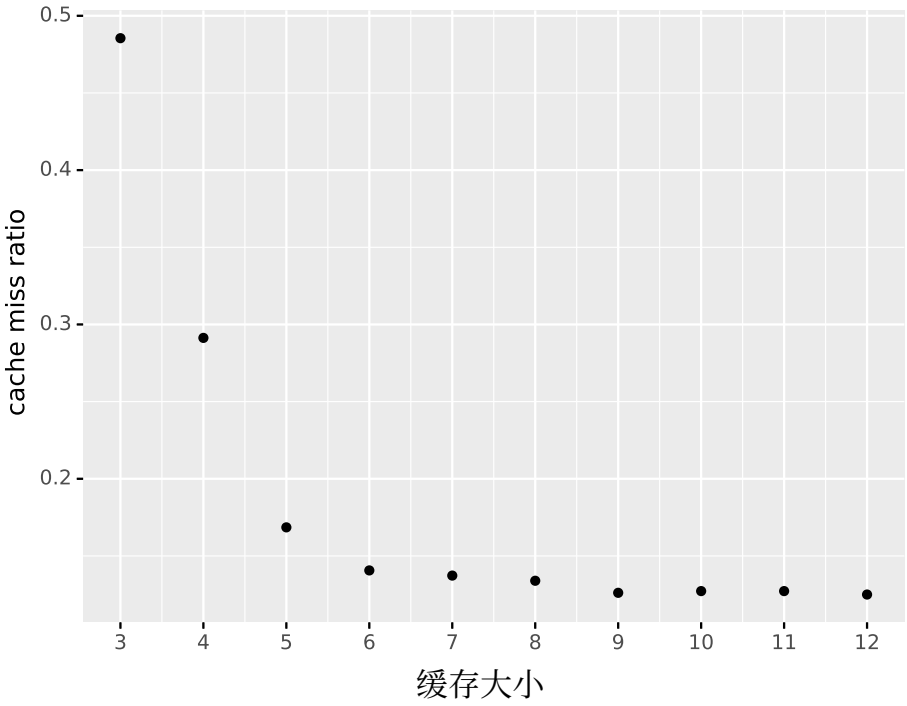


图7： 各运行器在不同缓存大小下的缓存未命中率。共有128个粒子在32个运行器上运行。为消除预热效应并排除记录不完整的周期，忽略了第一个和最后一个同化周期。

实验设置				
粒子	315	635	1,275	2,555
运行器数量	63	127	255	511
节点数量	64	128	256	512
模型进程	2,457	4,953	9,945	19,929
每个运行器的粒子数（平均）	5	5	5	5
粒子状态大小（GiB）	2.5	2.5	2.5	2.5
性能数据				
扩展效率	92%	91%	92%	87%
重采样（毫秒）	2.21	4.06	8.16	16.37
同化周期（秒）	136	138	139	146
传播（秒）	25.1	25.2	25.1	25.0
加载粒子状态 从并行文件系统加载到缓存（秒）	2.1	2.1	2.4	4.1
写入粒子状态 从缓存写入并行文件系统（秒）	1.4	1.6	1.8	2.3
每个周期写入并行文件系统的数据量（TiB）	0.77	1.55	3.11	6.24
每个周期从并行文件系统读取的数据量（TiB）	0.30-0.40	0.64-0.79	1.27-1.79	2.54-3.82

表1： 4 种不同规模下的实验设置和性能概览。所有情况下的时间均为平均值。模型时间步设为 100 秒。

相比之下，最长可达 6 秒）。在同化周期之间可以观察到一些常见的空闲时段，此时运行器正在等待某一周期的最后几次传播完成，以便服务器能够归一化权重、重新采样并再次开始分配工作。图 9 展示了这一现象，其中呈现了从某个运行器执行过程中记录的一段跟踪信息。该跟踪信息体现了运行器执行模拟实际任务（粒子传播和权重计算）的效率，而输入输出任务则被移至后台。

通过将粒子传播动态分配给可用的运行器，实现了计算任务的全局并行化。完全并行化的情况对应于 $R = P$，即运行器数量与粒子数相同。顺序执行的情况对应于 $R = 1$，即所有传播仅由一个运行器执行。不过，在这两个极限之间的取值可实现最佳并行效率。由于 WRF 传播粒子的时间变化性极低（最大变化幅度为 10%），当 R 能整除 P 时，我们观察到传播任务在运行器之间均匀分布（图8）。单次粒子传播耗时为 24 至 26.5 秒，

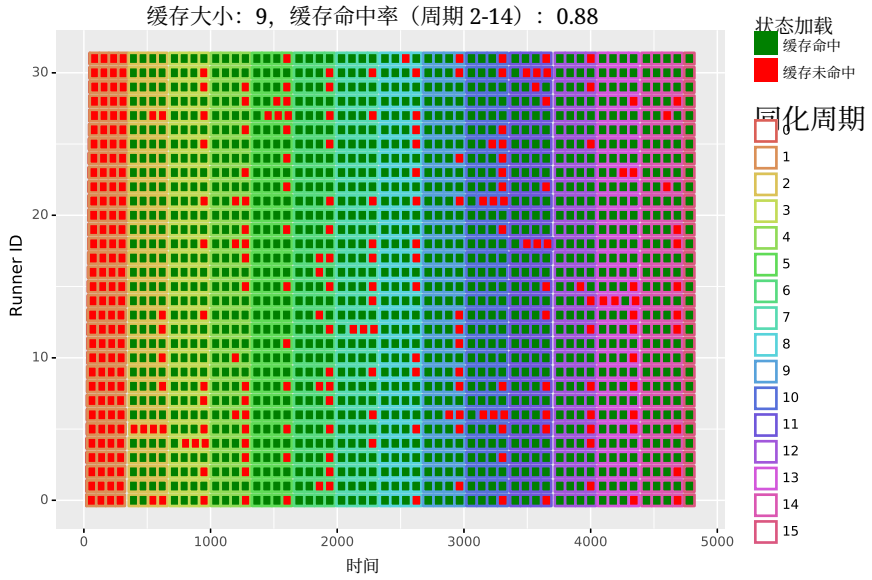


图8： 32 个运行器在 15 个同化周期中执行粒子传播的甘特图。如果相应的父粒子状态已存在于运行器缓存中，无需从 PFS 加载，则任务显示为绿色；否则显示为红色。

总体而言，这占平均同化周期的 87% 至 92%。权重计算耗时占 1%，与服务器通信耗时占 7% 至 12%，其中包括每个周期结束时的空闲时间（表 1—性能数据）。辅助进程、每个运行器节点占用的一个核心以及服务器（1 个节点）所需的额外资源，在我们规模最大的 512 个节点实验中仅占 2.7%。另一方面，利用运行器的粒子缓存以及服务器上的缓存感知动态调度，可将 $> 97\%$ 的状态加载完全转移至后台。否则，同步加载和写入粒子状态会使每次单粒子传播额外耗时约 6.4 秒，相当于平均传播时间的 14%（表1—*2,555* 个粒子）。

请注意，与传统的离线方法不同，我们只启动一次数值模拟代码，以执行多次粒子传播。初始化设置包括申请并分配运行器作业，以及初始化模拟代码。另一方面，传统的离线方法会将每次粒子传播分配给不同的作业，因此每个粒子都必须重新启动一次。启动

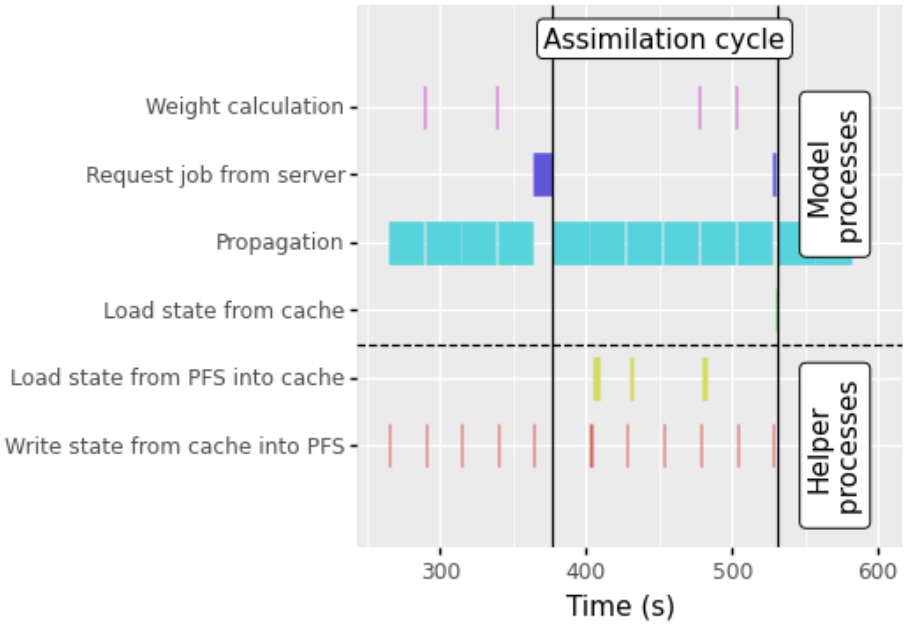


图 9： 详细展示运行器在一个同化周期内活动情况的轨迹。辅助进程使模型进程能够持续进行粒子传播；但在同化周期结束时，它们会等待服务器完成粒子重采样（深蓝色）。有些活动所占时间极短，因此在此处不可见（从缓存复制状态到模型进程）。

单个节点上的欧洲区域 WRF 模型，直到首次模型传播开始，需要 3–4 秒，不包括通过集群调度器分配作业所需的时间。

4.3. 服务器活动

我们进一步测量了服务器对运行器请求的响应能力。响应时间通常只有几百微秒，但部分作业请求耗时可达几秒（图 10）。不过，这些都是同化周期末尾出现的异常值，是负载均衡和粒子滤波器更新导致的空闲时间所致。在规模最大的实验中，使用 511 个运行器时，服务器在最大负载下每秒可处理 676 个请求。这表明，即使服务器运行的是顺序执行的 Python 代码，也足以支持这一规模。如果需要提升服务器速度，也可以进行简单优化（例如增加并行化）。

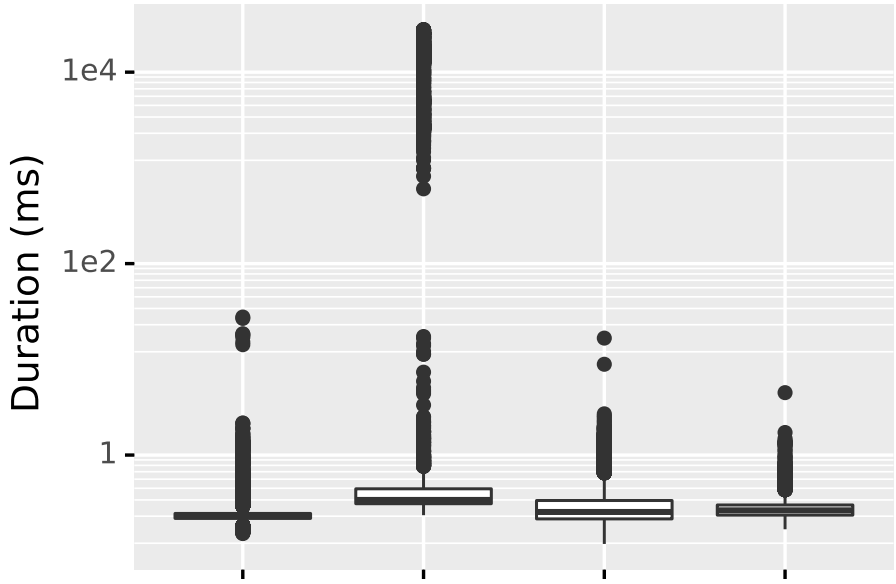


图 10： 运行器请求的服务器响应时间。

4.4. 与并行文件系统（*PFS*）之间的状态传输

接下来，我们仔细考察来自并行文件系统的粒子加载情况（图11）。样本量为 1024 个粒子、使用 256 个运行器且本地缓存大小为 9 个粒子时，每个周期会有 121 至 248 个粒子加载到缓存中。每个周期传播的不同父粒子数量 Q 介于 813 至 889 之间。每个父粒子最多传播 5 次，合计生成 1024 个粒子。与静态调度且不使用缓存时预期的 $Q + R - 1$ 上限相比，缓存显著减少了加载次数（公式15）。

对并行文件系统（PFS）的访问时间（加载/存储）变化显著，并且随着运行器数量的增加而增加（图12），这表明

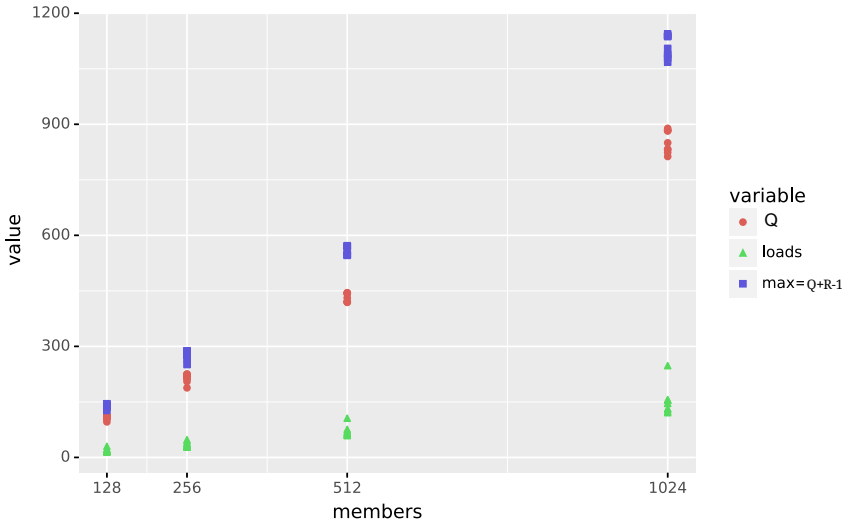


图11： 父粒子数量 Q、从并行文件系统加载到缓存的粒子数，以及不同集合大小下公式15给出的 $Q + R - 1$ 上界；缓存大小为9个粒子，每个运行器有4个粒子。

仅我们的应用程序就能给并行文件系统带来很大压力 [1]。每个粒子关联着 2.5 GiB 的数据。在每个同化周期中，为支持容错和动态负载均衡，所有已传播的粒子都会写入并行文件系统。对于我们在 512 个节点上使用 2,555 个粒子进行的实验，每个周期累计产生约 6.2 TiB 的数据（参见表 1）。然而，我们在 Jean-Zay 和 JUWELS 超级计算机上进行的实验表明，我们的框架以异步方式完成了大部分数据传输（第4.2节）。在不到 2% 的情况下，模型进程等待粒子加载的时间超过 0.1 秒；这些情况对应于辅助进程未能（完全）完成预取的情形。缓存中的本地加载和存储所需时间的平均值保持恒定，不受运行器数量影响（图13）。

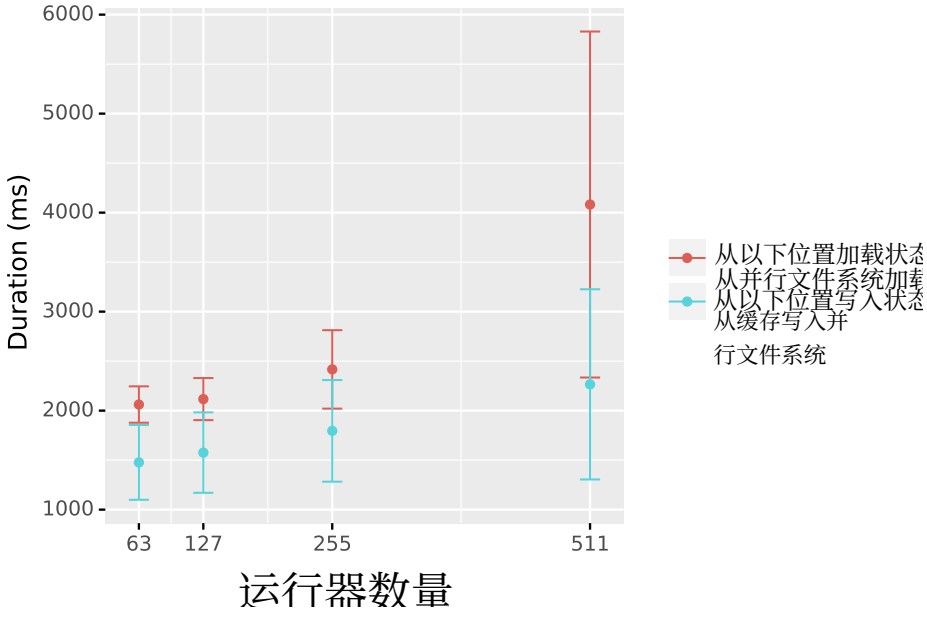


图12： 使用不同数量的运行器，将 2.5GiB 的粒子状态加载到并行文件系统或从并行文件系统存储所需的平均时间。

4.5. 容错、弹性与负载均衡

容错依赖于：1）将粒子持久化到并行文件系统（PFS）；2）框架的弹性，使其能够动态调整运行器数量。

[1]这些数值还可能受到集群上其他作业的影响

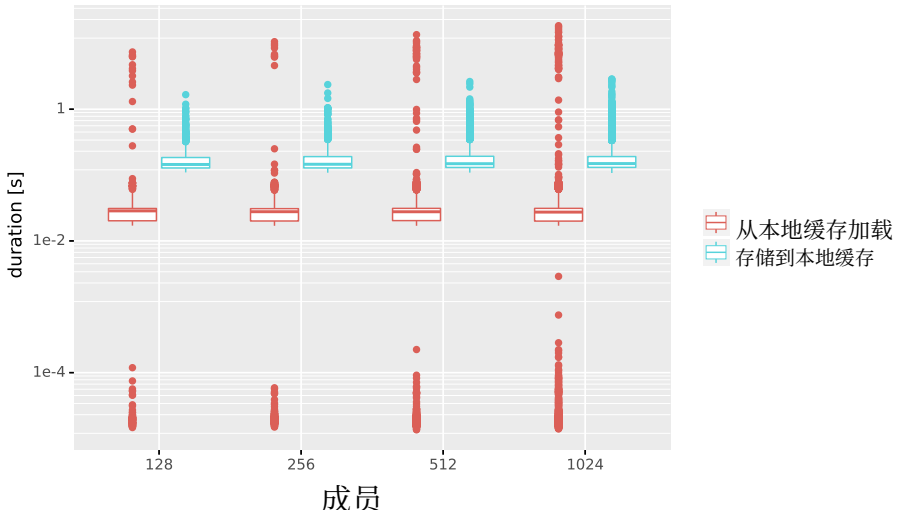


图13： 不同粒子数量下从本地缓存加载和存储所花费时间的箱线图。

如果某个运行器发生故障，启动器会请求执行一个新的运行器，以维持运行器数量不变。新的运行器连接到服务器后，会请求一个粒子并将其传播到服务器，该粒子由负载均衡算法分配。

我们在一项包含 63 个运行器的实验中测试了容错性和弹性，并人为导致 2 个运行器崩溃（图14）。首先请注意，容错算法能够做出恰当响应，在每次崩溃后都会重启一个新的运行器。第一次崩溃（运行器 #53）发生在最糟糕的情况下：恰逢当前周期正在传播最后一个粒子，导致出现显著的空闲时段。空闲时段首先是因为服务器需要等待超时（设为 60 秒），以确认运行器 #53 无响应；其次，除了运行器 #53 正在传播的那个粒子外，已无其他工作，而该粒子被重新分配给运行器 #44。与此同时，所有其他运行器都处于空闲状态，直到下一个周期开始。如果崩溃在周期中更早发生，空闲时段就会更短。第二次崩溃（运行器 #48）时便可观察到这一点，因为其他运行器仍在忙于传播工作。总体而言，我们观察到负载均衡效率较高，即使运行器数量发生变化，工作负载仍能在各运行器之间保持良好分布。

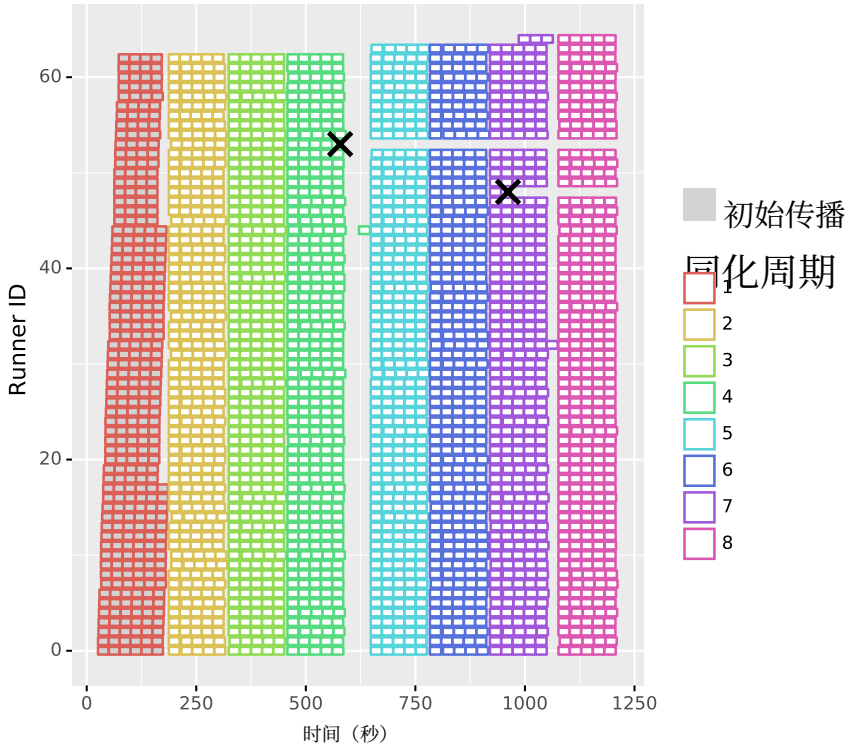


图14： 63 个运行器在 8 个同化周期中执行粒子传播的甘特图。运行器 #48 和 #53 崩溃（黑色叉号）后，两个新的运行器重新启动（顶部的两个运行器 #63 和 #64）。

4.6. 扩展性

我们评估了粒子滤波器在强扩展和弱扩展场景下的性能。强扩展场景中，运行器数量保持不变，同时增加粒子数；弱扩展场景中，粒子与运行器之比保持不变，同时增加运行器数量。在强扩展场景中，我们观察到，随着每个运行器的粒子数增加，运行器利用率呈上升趋势，并在约 96% 处达到平台期（图15）。由于异步预取几乎完全掩盖了全局输入/输出操作的耗时，增加每个运行器的粒子数主要有助于更好地分摊与重采样相关的同步开销。我们观察到同化周期的时间几乎保持不变，表明弱扩展表现理想。从 63 个运行器增加到 511 个运行器，周期耗时仅增加 8%，这表明扩展性表现高效。

将框架扩展至生产规模（图16）。在此规模下，使用 WRF 在欧洲区域进行粒子滤波，以开展短期天气预报，是对 Berndt 等人此前工作的重大推进 [21]。此外，除了以更高频率进行同化外，我们的方案还提供容错、自动负载均衡和弹性，同时最大限度地降低 I/O 成本和计算权重所需的时间。

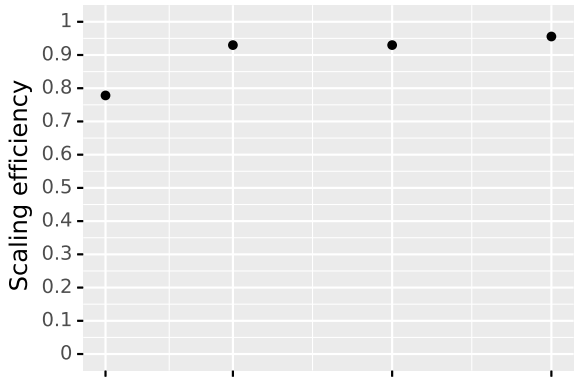


图15： 使用63个运行器和不同数量的粒子时的扩展效率。一个运行器设为参考案例。

4.7. 与基于文件的方法比较

	Melissa			ESIAS-met			ESIAS-met/Melissa
粒子	核心	时间（秒）	核心·秒/粒子	核心	时间（秒）	核秒/粒子。	资源使用比率
128	384	1062	3186	1536	267	3204	1.01
256	768	1062	3186	3072	317	3804	1.19
512	1536	1068	3204	6144	422	5064	1.58
1024	3072	1071	3213	12288	761	9132	2.84

表 2： 比较 Melissa 和 ESIAS-met（基于文件）运行时每个周期的资源使用量（核秒/粒子）。

我们将 Melissa 与基于文件的方法 ESIAS-met [22] 进行比较，使用相同的模拟代码 WRF（V3.7.1）和相同的数据集。对于相同的粒子数，两种方法使用的

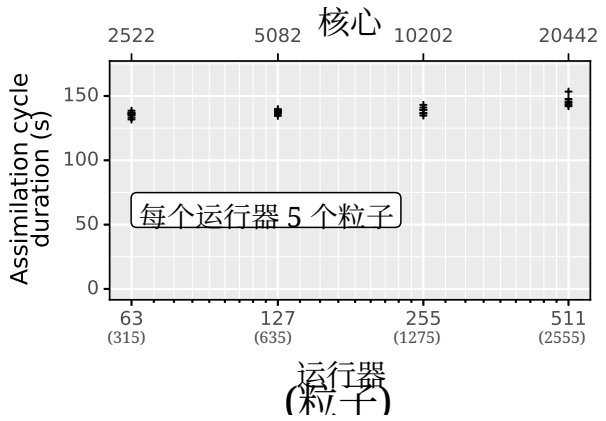


图16： 弱扩展性能测试：运行器数量不同时的同化周期时长，但每个运行器始终为 5 个粒子。

核心数差异很大（表 2）。在 ESIAS-met 中，每次粒子传播都需要启动一个专用的 WRF 实例。每次运行都要承担将粒子状态从文件加载并存储到文件的开销。对于 1024 个粒子，ESIAS-met 使用 12288 个核心，而 Melissa 只需 3072 个核心，因为运行器可分别传播多个粒子。因此，ESIAS-met 的执行时间更短，因为它高度并行化；但得益于为提高效率而开发的综合策略，Melissa 的资源使用量（核秒/粒子/周期）改善了 2.84 倍。随着粒子数增加，收益也随之提升，这表明在目标集合规模较大时，Melissa 方法尤其有利。

5. 讨论

在第 4.1 节中，我们推导出，在欧洲区域使用 2,555 个粒子进行 48 个时间步的粒子滤波，产生的数据总量约为 300 TiB。后处理如此大量的数据颇具挑战。我们可以按照 Terraz 等人提出的方法，利用原位数据处理技术扩展我们的框架 [23]。

我们只考虑了传播时间长于从并行文件系统加载状态所需时间的情况；而对于传播时间短于状态加载时间的应用程序，我们的方案效率会受到限制，因为在这种情况下，我们无法进一步隐藏 I/O 成本。另一方面，在 WRF 场景中，由于我们进行每小时重采样，传播时间已经很短，因为我们每

小时进行重采样。我们选择这一频率，主要是为了对我们的方案进行压力测试。生产运行通常不需要如此高的频率，而且传播时间往往更长，正如我们的实验所示。不过，为了进一步缩短传输时间，我们正在评估利用节点本地持久存储作为全局共享存储层的方法。此类解决方案已 readily available，形式包括分布式临时文件系统 [24] ，例如 BeeOND、GekkoFS 和 BurstFS。我们还试验了将运行器相互连接以建立点对点网络，使运行器能够彼此直接交换所需的状态。

在我们使用 WRF 进行的实验中，粒子传播时间相对均匀，波动幅度最多为 10%。在 WRF 中采用不同的物理过程、使用其他模拟代码，或者运行器运行在异构资源上时（例如某些运行器利用图形处理器进行传播，速度快于其他运行器），都可能出现波动更大的情况。此外，来自其他情境的用例，例如基于模拟的推断（SIB）和集成分类，也可以通过我们的框架实现，并可能导致传播时间大不相同。因此，在这些条件下测试我们的框架是未来一项重要的工作。

我们目前提出的方法依赖于不会计算内部成员状态修正的滤波器。要将我们的方法扩展到这类粒子滤波器 [1]，可能需要向服务器汇总的不只是粒子权重。未来，我们计划实现所提出的粒子滤波器，并探索使框架适用于这类情形所需满足的条件。我们使用 SIR粒子滤波器验证了所提出的方法，但还有许多变体，这些变体也是当前活跃的研究课题 [25, 26]。其中一个挑战是，所需粒子数会随着问题维度呈指数级增长 [27, 28]。对于地球科学用例而言，这一问题尤为突出，因为正如本文所述，这类用例涉及高维空间。综述 [1] 全面介绍了地球科学中使用粒子滤波器进行数据同化的方法，以及应对维数问题的途径。

本文所用的粒子滤波器要求在每个同化周期结束时设置一个同步点。对于我们的框架而言，这是目前效率低下的主要原因。若要放宽这一要求，就需要重新审视粒子滤波算法；这也是当前的一个研究热点 [29, 30, 31]。

6. 相关工作

数据同化领域涵盖了种类繁多的技术和算法，例如松弛逼近 [32], 克里金法 [33], 集合卡尔曼滤波 [34], 集合最大

似然滤波器 [35], 或粒子滤波器 [36]。概述请参阅 [37, 38]。本文聚焦于统计数据同化，它依赖模型的一次集合运行来计算统计估计量（集合卡尔曼滤波中的协方差矩阵，粒子滤波器中的概率密度函数）。

为汇总所有成员（即粒子）产生的数据，主要采用两类方法：要么将数据存储到文件中，再在第二步进行处理（离线模式）；要么在线处理数据，通常是在负责运行成员并处理数据的大型 MPI 代码中进行。采用离线模式的框架包括 EnTK [39], ；已发表的数据同化应用案例中规模最大的一项，是使用 EnKF 滤波器的分子动力学应用，共有 4,096 个成员 [40]。OpenDA 也采用这种模式，通过 NetCDF 与 NEMO 代码交换数据 [41]。DART 同时支持这两种模式 [42], ；文献报告了采用离线模式的大规模数据同化案例： [43]（海洋代码，约 1,000 个成员），以及 [44, 45] （1,024 个成员、局地集合变换卡尔曼滤波、富岳上的 600 万个核心）。基于文件的方法具有简单易用的优点，并能提供容错能力和弹性。但这些方案不支持成员虚拟化、状态缓存和预取。因此，启动或重启一个成员都需要申请新的资源分配，并启动模型代码的新实例，同时承担所有相关的启动开销。节点本地持久存储功能（例如使用固态硬盘）可以存储中间文件，从而减轻并行文件系统的 I/O 瓶颈。在 [44], 中，这类存储用于存储成员状态，但没有专门的容错机制。因此，如果节点发生故障，导致节点本地存储不可用，就需要重新计算丢失的成员状态。除了利用节点存储构建分布式缓存之外，我们未来的目标之一是使用节点存储，而非并行文件系统，作为全局共享文件系统层。

在线模式避免了 I/O 瓶颈。PDAF [46], 支持这两种模式，例如曾以在线方式将观测值同化到区域地球系统模型 TerrSysMP 中。数据同化基于集合卡尔曼滤波，集合成员最多达 256 个 [47]。ESIAS 在欧洲的风电模拟中通过粒子滤波器进行在线数据同化，粒子最多达 4,096 个 [21]。请注意，本文使用的 ESIAS 中包含相同的 WRF 组件，采用的配置具有相似的计算区域，但空间分辨率更高，物理过程也更先进、计算耗时更长。我们还发现了一些用于在线数据同化的专用消息传递接口代码，如 [48] （大气模型、10,240 个集合成员、局地集合卡尔曼滤波器、4,608 个计算节点）。然而，所有这些消息传递接口方案都会形成单体代码，且不支持容错、弹性或负载均衡。在 [49], 中，作者分析了各种粒子传播调度

方法，但规模有限（6 个计算节点和 300 个粒子）。我们在与所提出方案架构相似的环境中进行了实验，但采用的是集合卡尔曼滤波，而非粒子滤波 [19]。我们证明了容错性、弹性和可扩展性：在使用 Parflow 水文代码进行集合卡尔曼滤波数据同化的实验中，集合成员最多达 16 k，核心数最多达 16 k。与我们提出的新方案不同，集合卡尔曼滤波需要集中式滤波器更新，将完整的状态集合汇集到中心实例，以进行观测同化。在我们提出的粒子滤波新方案中，我们利用粒子滤波器的某些特性来消除服务器瓶颈，并显著减少数据移动。

7. 结论

本文提出了一种用于处理粒子滤波器超大规模集合的架构。该架构旨在应对百亿亿次级计算的挑战，从而支持大规模集合运行 [50]。该架构采用服务器/运行器模型：运行器支持分布式缓存和粒子传播虚拟化，而服务器则汇总运行器计算出的权重，并确保工作负载的动态负载均衡。粒子传播经过虚拟化，因此所需的运行器数量不受粒子数限制。通过加入分布式检查点机制，该架构支持在执行期间动态调整运行器数量，以实现容错和弹性。使用 WRF 天气模拟代码开展的实验表明，我们的框架能够在 20,442 个核心上运行至少 2,555 个粒子，扩展效率达到 87%。动态粒子传播调度和缓存使全局输入/输出操作减少了 88%。与 ESIAS 基于文件的方法相比，Melissa 在 1024 个粒子时的资源使用效率提升了 2.83 倍。

未来的工作包括试验自适应或局部化粒子滤波器，以及结合粒子滤波器与卡尔曼滤波器。我们还计划扩展分布式缓存和容错算法，以完全避免使用集中式文件系统，并仅依赖节点本地固态硬盘存储粒子。

致谢

本项目获得欧盟“地平线2020”研究与创新计划资助，资助协议编号为 824158（EoCoE-2）。本研究获准使用IDRIS的高性能计算资源

使用GENCI（国家高性能计算大型设备）授予的分配项目2020-A8 A0080610366。作者衷心感谢高斯超级计算中心（ www.gauss-centre.eu）为本项目提供资助，并通过约翰·冯·诺依曼计算研究所（NIC）在J¨ulich超级计算中心（JSC）的GCS超级计算机 JUWELS上提供计算时间。我们感谢能够获取JSC的SDL气候科学 Meteocloud提供的气象输入数据。我们还感谢PRACE授予我们使用德国 J¨ulich超级计算中心（JSC）JUWELS的权限。

参考文献

[1] P. J. Van Leeuwen、H. R. K¨unsch、L. Nerger、R. Potthast、S. Reich，《高维地球科学应用中的粒子滤波器：综述》，《皇家气象学会季刊》145 (723) (2019) 2335–2365。[2] D. Lund´en、J. Borgstr¨om、D. Broman，《概率编程语言中序贯蒙特卡洛推断的正确性》，载于：ESOP，2021年，第404–431页。[3]F. Ronquist、J. Kudlicka、V. Senderov、J. Borgstr¨om、N. Lartillot、D. Lund´en、L. Murray、T. B. Sch¨on、D. Broman，《通用概率编程为统计系统发育学提供了一种强有力的方法》，《通讯生物学》4 (1) (2021) 1–10。[4] J.-W. van de Meent、B. Paige、H. Yang、F. Wood，《概率编程导论》，arXiv，2018年。 doi:10.48550/ARXIV.1809.10756。网址：https://arxiv.org/abs/1809.10756[5] J. F. G. de Freitas、M. Niranjan、A. H. Gee、A. Doucet，《用于训练神经网络模型的序贯蒙特卡洛方法》，《神经计算》12 (4) (2000年) 955–993。doi:10.1162/089976600300015664。[6] P. M. Blok、K. van Boheemen、F. K. van Evert、J. IJsselmuiden、G.-H. Kim，《基于粒子滤波器和卡尔曼滤波器定位的果园机器人导航》，《农业中的计算机与电子学》157 (2019年) 261–269。doi:10.1016/j.compag.2018.12.046。

网址 https://www.sciencedirect.com/science/article/pii/S0168169918315230[7] P. Bauer、P. D. Dueben、T. Hoefler、T. Quintino、T. C. Schulthess、N. P. Wedi，《地球系统科学的数字革命》，《自然计算科学》1 (2) (2021年) 104–113。[8] W. C. Skamarock、J. B. Klemp、J. Dudhia、D. O. Gill、D. Barker、M. G. Duda、J. G. Powers，《先进研究 WRF 第3版说明》，技术报告编号 NCAR/TN-475+STR，美国大气研究大学联盟（2008年）。[9] J. V. Candy，《自举粒子滤波》，《IEEE 信号处理杂志》24 (4) (2007年) 73–85，会议名称：IEEE Signal Processing Magazine。 doi:10.1109/MSP.2007.4286566。[10] N. Gordon、D. Salmond、A. Smith，《非线性/非高斯贝叶斯状态估计的新方法》，《IEE F辑：雷达与信号处理》140 (2) (1993年) 107。 doi:10.1049/ip-f-2.1993.0015。网址 https://digital-library.theiet.org/content/journals/10.1049/ip-f-2.1993.0015[11] J. S. Liu、R. Chen、T. Logvinenko，《带重采样的序贯重要性抽样理论框架》，载于：A. Doucet、N. de Freitas、N. Gordon（编），《实践中的序贯蒙特卡洛方法》，《工程与信息科学统计》，施普林格，纽约，纽约州，2001年，第225–246页。 doi:10.1007/978-1-4757-3437-9_11。网址 https://doi.org/10.1007/978-1-4757-3437-9_11[12] M. Bolic、P. Djuric、Sangjin Hong，《粒子滤波器的新重采样算法》，载于：2003年IEEE声学、语音与信号处理国际会议论文集（ICASSP '03），第2卷，IEEE，香港，中国，2003年，第II–589–92页。 doi:10.1109/ICASSP.2003.1202435。网址 http://ieeexplore.ieee.org/document/1202435/[13] R. L. Graham，《某些多处理器异常的界限》，《贝尔系统技术期刊》45 (9) (1966年) 1563–1581。[14] D. Shmoys、J. Wein、D. Williamson，《并行机器的在线调度》，载于： [1991]第32届年度基础研讨会论文集

计算机科学，IEEE计算机学会出版社，波多黎各圣胡安，1991年，第131–140页。[15] Slurm工作负载管理器— https://slurm.schedmd.com/。[16] OAR资源与任务管理器— http://oar.imag.fr/。[17] A. Merzky、M. Turilli、M. Titov、A. Al-Saadi、S. Jha，《Radical-Pilot在领导级平台上的设计与性能特征分析》（2021年）。 doi:10.48550/ARXIV.2103.00091。网址：https://arxiv.org/abs/2103.00091[18] QCG-PilotJob— https://github.com/psnc-qcg/QCG-PilotJob。[19] S. Friedemann、B. Raffin，《一种用于基于集合的大规模资料同化的弹性框架》，《国际高性能计算应用期刊》36 (4) (2022) 543–563。网址：https://hal.inria.fr/hal-03017033[20]L. Bautista-Gomez、S. Tsuboi、D. Komatitsch、F. Cappello、N. Maruyama、S. Matsuoka，《FTI：混合系统的高性能容错接口》，载于：SC '11：2011年国际高性能计算、网络、存储与分析会议论文集，2011年，第1–12页。[21] J. Berndt，《风电预测中异常误差事件的可预测性——一种超大规模集合方法》，博士论文，科隆大学（2018年）。[22]Y.-S. Lu、G. Good、H. Elbern，《利用大气模拟随机积分集合的气象组件1.0版优化欧洲区域云量天气预报》，第1–31页。出版方：Copernicus GmbH。doi:10.5194/gmd-2022-118。网址：https://gmd.copernicus.org/preprints/gmd-2022-118/#discussion [23] T. Terraz、A. Ribes、Y. Fournier、B. Iooss、B. Raffin，《Melissa：避免中间文件的大规模传输过程中的敏感性分析》，载于：国际高性能计算、网络、存储与分析会议（SC'17），丹佛，2017年。

[24] A. Brinkmann、K. Mohror、W. Yu、P. Carns、T. Cortes、S. A. Klasky、A. Miranda、F.-J. Pfreundt、R. B. Ross、M.-A. Vef, 《面向高性能计算的临时文件系统》, 《计算机科学与技术杂志》35 (1) (2020年) 4–26。 doi:10.1007/s11390-020-9801-1。网址: https://doi.org/10.1007/s11390-020-9801-1[25] A. Doucet、S. Godsill、C. Andrieu, 《关于贝叶斯滤波的序贯蒙特卡洛采样方法》, 《统计与计算》10 (3) (2000年) 197–208。[26] S. C. Surace、A. Kutschireiter、J.-P. Pfister, 《如何避免维数灾难：有无重要性权重时粒子滤波器的可扩展性》, 《SIAM评论》61 (1) (2019年) 79–91。[27] C. Snyder、T. Bengtsson、M. Morzfeld, 《使用最优提议分布的粒子滤波器性能界》, 《月度天气评论》143 (2015年) 12。[28] P. Fearnhead、H. Kunsch, 《粒子滤波器与资料同化》, 《统计学及其应用年度评论》5 (1) (2018年) 421–449，arXiv：1709.04196。[29] C. Verge、C. Dubarry、P. Del Moral、E. Moulines, 《序贯蒙特卡洛方法的并行实现：岛屿粒子模型》, 《统计与计算》25 (2) (2015年) 243–260。[30] A. Jasra、A. Lee、C. Yau、X. Zhang, 《存活粒子滤波器》, arXiv预印本 arXiv:1304.0151。[31] V. Elvira、J. Míguez、P. M. Djurić, 《通过在线收敛评估方案调整序贯蒙特卡洛方法中的粒子数》, 《IEEE信号处理汇刊》65 (7) (2016年) 1781–1794。[32] V. R. N. Pauwels、R. Hoeben、N. E. C. Verhoest、F. P. De Troch, 《通过资料同化利用遥感土壤湿度空间分布模式改进小尺度流域流量预测的重要性》, 《水文学杂志》251 (1) (2001年) 88–102。

[33] J. L. Williams, R. M. Maxwell，利用完全耦合的随机模拟将地下不确定性传播至大气，《水文气象学杂志》12 (4) (2011) 690–701。[34] D. Zhang, H. Madsen, M. E. Ridler, J. Kidmose, K. H. Jensen, J. C. Refsgaard，土壤湿度和地下水的多变量水文数据同化，Hydrol. Earth Syst. Sci. 20 (10) (2016) 4341–4357。[35] S. Q. Zhang, M. Zupanski, A. Y. Hou, X. Lin, S. H. Cheung，在云分辨率WRF集合数据同化系统中同化受降水影响的辐射，《月度天气评论》141 (2) (2013) 754–772。[36] van Leeuwen, J. P.，一种适用于大规模应用的方差最小化滤波器，Mon. Wea. Rev. 131 (9) (2003) 2071–2084。[37] M. Asch, M. Bocquet, M. Nodet，数据同化：方法、算法与应用，第11卷，SIAM，2016。[38] G. Evensen，数据同化：集合卡尔曼滤波器，Springer Science & Business Media，2009。[39] V. Balasubramanian, M. Turilli, W. Hu, M. Lefebvre, W. Lei, G. Cervone, J. Tromp, S. Jha，发挥众人之力：面向可扩展集合应用的可扩展工具包，载于：IPDPS 2018，2018。[40] V. Balasubramanian, T. Jensen, M. Turilli, P. Kasson, M. Shirts, S. Jha，大规模自适应集合生物分子应用，《SN计算机科学》1 (2) (2020) 1–15。[41] N. van Velzen, M. U. Altaf, M. Verlaan，用于海洋数据同化的OpenDA-NEMO框架，《海洋动力学》66 (5) (2016) 691–702。[42] J. Anderson, T. Hoar, K. Raeder, H. Liu, N. Collins, R. Torn, A. Avellano，数据同化研究试验平台：一项社区设施，《美国气象学会通报》90 (9) (2009) 1283–1296，出版方：美国气象学会。[43] H. Toye, S. Kortas, P. Zhan, I. Hoteit，面向大规模业务化集合数据同化的容错高性能计算调度器扩展：红海应用，《计算科学杂志》27 (2018) 46–56。

[44] H. Yashiro, K. Terasaki, Y. Kawai, S. Kudo, T. Miyoshi, T. Imamura, K. Minami, H. Inoue, T. Nishiki, T. Saji, M. Satoh, H. Tomita，采用3.5公里网格全球天气模拟的1024成员集合资料同化，载于：Supercomputing 2020：国际高性能计算、网络、存储与分析会议（SC），IEEE计算机学会，洛斯阿拉米托斯，加利福尼亚州，美国，2020年，第1–10页。[45] H. Yashiro, K. Terasaki, T. Miyoshi, H. Tomita，集合资料同化吞吐量感知框架的性能评估：nicam-letkf案例，《地球科学模型发展》9 (7)。[46]L. Nerger, W. Hiller，基于集合的资料同化系统软件——实现策略与可扩展性，Computers & Geosciences 55 (2013) 110–118。[47] W. Kurtz, G. He, S. J. Kollet, R. M. Maxwell, H. Vereecken, H.-J. Hendricks Franssen，TerrSysMP–PDAF（1.0版）：用于陆面—地下综合模型的模块化高性能数据同化框架，《地球科学模型发展》9 (4) (2016) 1341–1360。[48]T. Miyoshi, K. Kondo, T. Imamura，使用中间大气环流模式进行10,240成员集合卡尔曼滤波：使用大气环流模式的10,240成员集合卡尔曼滤波，《地球物理研究快报》41 (14) (2014) 5264–5271。[49] F. Bai, F. Gu, X. Hu, S. Guo，大规模时空系统中分布式粒子滤波器的粒子路由，《IEEE并行与分布式系统汇刊》27 (2) (2015) 481–493。[50] T. C. Schulthess, P. Bauer, N. Wedi, O. Fuhrer, T. Hoefler, C. Schar，思考百亿亿次级计算的目标与基准：基于天气与气候模拟的路线图，《科学与工程计算》21 (1) (2019) 30–41。