将计算带到数据所在之处：一种由多目标进化算法驱动的方法，用于优化平方公里阵列和SRCNet背景下的数据处理

´Manuel Parra-Roy´on、Alvaro Rodr´ıguez-Gallardo、Susana S´ánchez-Exp´osito、Laura Darriba-Pol、Jes´ús S´ánchez-Casta˜neda、´MAngeles Mendoza、Juli´án Garrido、Javier Mold´ón 和 Lourdes Verdes-Montenegro，安达卢西亚天体物理研究所，IAA-CSIC，Glorieta delaAstronom´ía，s/n，18018，´格拉纳达，西班牙；电子邮箱：{mparra、jmoldon、arodriguez}@iaa.es

arXiv:2601.01980v1 [cs.DC] 5 Jan 2026

摘要—**平方公里阵列（SKA）是一个由下一代射电天文学驱动的大数据设施，将彻底改变我们对宇宙及基础物理学规律的理解，并需要创新的高效数据处理解决方案。SKA区域中心网络（SRCNet）是一个协作生态系统，承担着处理和分析SKA数据产品这一艰巨任务。对于SKA而言，近百亿亿次级计算将构成一项挑战，其中最主要的问题是数据移动。随着计算能力不断提升，所生成数据的庞大规模也变得惊人。由于现有网络和存储基础设施的限制，将海量数据移动至集中式计算资源的传统方法变得不可行。数据传输瓶颈成为关键障碍，阻碍了整体效率的提升。要克服这一挑战，亟需进行范式转变。原位处理和分布式计算模型等策略应运而生：在这些模型中，计算被带到数据所在之处，被视为颇具前景的解决方案。**

在平方公里阵列（SKA）领域，尤其是在 SRCNet 的数据处理需求中，将函数即服务（FaaS）与由进化算法（EAs）驱动的决策实体相结合至关重要。FaaS 抽象化了基础设施管理方面的工作，使模块化函数能够部署在靠近数据源的位置。这一发展符合将计算带到数据所在位置的原则，从而缓解大规模数据传输所带来的挑战。在进化算法指导下，决策实体能够系统地探索近似最优的执行计划，并提供有关函数应如何以及在何处于整体计算与数据基础设施中执行的详细信息。以执行时间和能耗为两个目标，同时考虑数据传输或数据位置等约束，多目标进化算法（MOEAs）可以成为优化计算向数据移动的良好选择。在此背景下，MOEAs 可以为 SRCNet 内计算模型的高效且经济的数据处理提供基准指导。我们的方案将 FaaS 部署的技术层面，与定制的首个多目标进化算法模型的数学建模和代码实现相结合，用于优化 SRCNet 架构内的函数执行计划，并实现其与 FaaS 的集成。

I. 引言

平方公里阵列望远镜将在灵敏度、分辨率和巡天速度方面实现显著提升。由两个主要阵列——分别位于南非和西澳大利亚的SKA中频阵列与SKA低频阵列——组成的平方公里阵列将能够覆盖从50 MHz到14 GHz的宽广频率范围。平方公里阵列望远镜的科学目标广泛而雄心勃勃，涵盖从宇宙黎明到生命起源的研究，同时也包括探索性研究。由于需要向全球各地的终端用户传输、处理、存储和分发海量数据，平方公里阵列项目被许多人视为终极大数据挑战。

平方公里阵列天文台（SKAO）将由遍布全球的平方公里阵列区域中心网络（SRCNet）提供支持。SRCNet [1] 将在推动天文学研究方面发挥关键作用，满足天文学家对数据处理能力的需求。SRCNet 将经过战略性设计，以高效归档、分发和处理 SKAO 望远镜生成的大量科学数据产品。平方公里阵列数据挑战[2], [3] 等方法有助于了解 SRCNet 预期的数据量和计算类型。

SRCNet 内部的计算模型尚未确定。目前已提出多种方案，包括云环境、分布式系统或数据网格方案。科学用户将向 SRCNet 提交各种各样的计算流程和任务，从运行在工作负载管理器（例如 Slurm）之上的复杂流水线，到专注于使用大量数据的独立操作和函数。望远镜产生的数据将分布在 SRCNet 的各个节点中，因此有必要考虑数据和计算所在的位置，使待运行的函数尽可能靠近数据。该模型定义了一种数据驱动架构，其中计算会移动到数据所在的位置 [4]。图 1 展示了一个高度简化的 SRCNet 架构，其中每个节点都为系统贡献 CPU、GPU 及其他计算元素。此外，每个节点还存储不同的数据集合；这些数据集合可能在不同位置或节点上可用，也可能不可用，并部署了一组与计算能力相匹配的软件函数。网络

与存储一起，也在确定系统约束的部分内容方面发挥着重要作用。

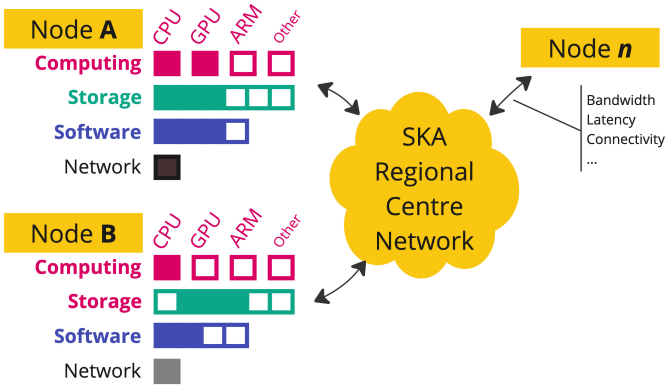

图 1. 简化 SRCNet 结构示意图，展示了不同互连节点所提供的不同计算元素、存储容量、软件/服务可用性和网络特性。每个区块代表一组元素，例如，对于计算而言，CPU 可表示可用核心的数量；存储表示可用的数据集合；软件表示可用于处理的软件；网络则表示与其他节点的连接性。

这种架构需要了解 SRCNet 的相关信息，例如数据的位置、访问数据和使用计算资源的权限、计算任务的需求，以及软件和硬件的可用情况等。还应考虑 SRCNet 中的其他参数，例如节点间的带宽或延迟。此外，该架构按照每个建议执行的函数的规范运行。

函数即服务（FaaS）为云计算（CC） [5] 带来了变革性范式，能够有效降低传统计算模式所伴随的巨额开销。通过将基础设施管理任务抽象化，FaaS[6] 让开发者能够独立部署单个函数，无需持续进行资源分配。这种精简的方法大幅减少了时间开销，使代码能够直接在数据上快速、高效地执行。开发者编写特定函数，然后将其部署到云计算提供商的无服务器环境中。这种方法免去了基础设施管理的需要，支持按需付费模式，并提升可扩展性和资源分配效率。尽管 FaaS 无法解决所有问题，但它提供了一个初步方案，可支持在分布式数据上执行任务/函数。就我们的情况而言，可以将 SRCNet 问题简化为一种模型：用户调用需要 CPU 或 GPU 等计算资源、并作用于 SRCNet 内分布式数据的单个函数。这些函数可以容器化、部署并通过函数即服务提供。Kubernetes 等平台为此提供支持，使得计算资源能够分配给特定函数等功能成为可能，从而提供出色的粒度，以便支持可高度扩展的工作负载。

在这项工作中，我们提出了一种基于多目标进化算法的模型设计，以尽可能高效、优化的方式应对将计算带到分布式节点网络的挑战；这些节点存储着数据集合，设计还会考虑节点在计算单元（CPU、图形处理器等）、存储或连接性等方面的能力。该模型聚焦于两个目标：a) 资源消耗/使用，b) 操作的能源效率。其基本思路是制定执行计划，以确定如何对 SRCNet 科学用户所需的数据执行任务/函数。该方案将系统状态纳入考量，动态为多目标进化算法提供输入，并生成能够适应 SRCNet 全局执行需求情境的计划。

本文结构如下：第 2 节总结相关工作；随后，第 3 节阐述所部署的函数即服务层的技术细节，该层将作为执行计划的编排设施。第 4 节定义数学模型并介绍多目标进化算法的实现。最后，第 5 节讨论所提出模型的结论。

II. 相关工作

许多科学研究探讨了分布式系统中与计算和效率相关的挑战与机遇 [7]。文献 [8] 讨论了在分布式节点间优化计算任务的复杂性，并强调了智能资源分配的必要性。效率方面的考量不仅限于计算速度，还包括负载均衡、容错能力以及对不同工作负载的适应性。文献 [9] 通过优化任务分配和资源利用，为实现分布式系统中的高效率提供了重要见解，并为我们在 SRCNet 架构背景下提升计算效率的方法奠定了基础。

在数据网格范式的背景下，人们也探索了分布式系统中数据的高效管理与处理 [10]。数据网格倡导去中心化的数据架构方法，这与将计算带到数据所在位置的理念一致。数据域的去中心化，以及将数据产品建立为可独立部署的单元，都契合优化数据移动和提升整体系统效率的目标。这一范式补充了我们的策略，助力在 SRCNet 中编排分布式数据上的操作执行。

函数即服务的兴起彻底改变了云计算范式。 [5]和 [11] 等研究探讨了函数即服务的原理与应用，凸显了其在抽象基础设施复杂性方面的作用。函数即服务支持模块化函数的无缝部署与执行，改变了开发者与云计算资源交互的方式。我们的工作与这些研究一脉相承，将函数即服务作为执行层，用于编排 SRCNet 中云计算资源的利用。

多目标进化算法在应对优化挑战方面日益受到重视，尤其是在计算与

效率方面。 [12] 的研究展示了多目标进化算法在分布式系统中寻找资源分配和能源效率最优解方面的有效性。我们的工作以此为基础，采用多目标进化算法的方法，动态优化 SRCNet 中分布式数据操作的执行计划，在资源消耗和能源效率等目标之间取得平衡。

FaaS 与多目标进化算法的结合，为优化云计算资源提供了一种新颖的方法。本文探讨了无服务器计算与进化算法如何协同工作，以支持资源高效型应用。FaaS 具有敏捷性和可扩展性，而多目标进化算法则提供了一种系统化的方法来平衡相互冲突的目标。我们提出的方法利用这种协同效应，将 FaaS 用作执行编排层，并使用多目标进化算法动态调整执行计划。

下一节将概述 FaaS 层的架构和运行方式，该层充当执行计划的动态编排机制。

III. 函数即服务部署

本节探讨 FaaS 的部署方式，即在 Kubernetes 之上使用 OSCAR 平台。OSCAR 专为 Kubernetes 集群上基于 Docker 的计算任务而设计，可部署于多云环境，因此是实现灵活且可扩展的 FaaS 部署的理想选择。OSCAR 集群由多个组件构成，包括用于弹性管理的 CLUES、用于高性能分布式对象存储的 MinIO、作为无服务器平台的 Knative、用于 API 和服务管理的 OSCAR Manager，以及面向终端用户的图形界面 OSCARUI，如图 2 所示。在存储方面，OSCAR 支持多种外部存储提供商，包括 MinIO 服务器、Amazon S3、EGI 联邦云中的数据访问，以及任何 WebDAV 存储提供商，例如 dCache 或 StoRM。

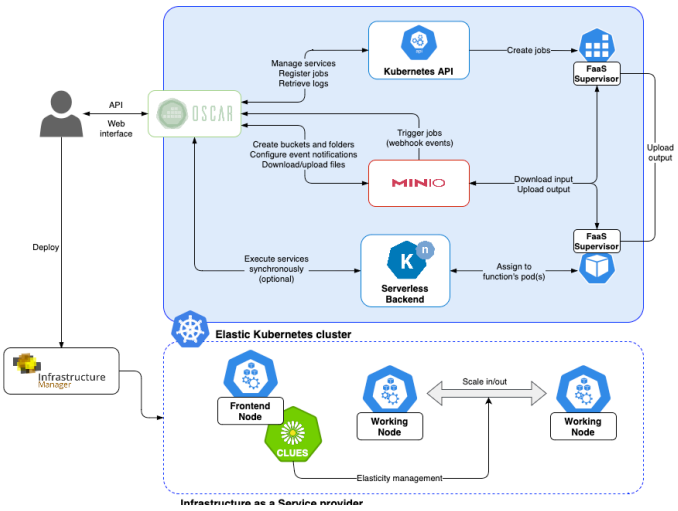

图 2. OSCAR 和函数即服务组件，包括 KNative 集成。

我们选择了 OSCAR，因为它作为 FaaS 服务与 KNative 配合使用。OSCAR 也专为数据处理而设计，因为它通过数据事件触发并行函数调用，以进行文件处理。此外，KNative 提供可自定义的代理器功能。该代理器提供可发现的事件输入端点（系统详细信息），并使用触发器传递事件（执行函数）。代理器充当计算节点资源池与在各个节点上执行函数能力之间的智能中介。其主要功能是动态优化这些函数/任务在特定节点上的分配，并根据系统当前状态及其他变量做出明智决策。代理器收集这些系统状态信息，并针对每个节点考虑：a) 计算单元（CPU、GPU、ARM 等）的可用性和使用情况；b) 数据集合的访问权限和可用性；以及 c) 节点之间访问和连接带宽方面的系统指标。利用所有这些信息，代理器可以使用优化算法，更智能地决定如何更高效地执行计算任务。

接下来的小节将概述包含基本组件的 OSCAR 安装流程，然后介绍如何将函数容器化并在函数即服务和 KNative 中发布。

A. 使用 *OSCAR* 部署 *OSCAR* 并在函数即服务中发布函数

OSCAR 的安装流程包括以下步骤：a) 安装 Kubernetes 集群，部署 Ingres-Nginx，在所有节点上部署 MInIO 和网络文件系统，部署 KNative（用于函数即服务），最后部署 OSCAR 组件。所有步骤及更详细的说明均可在项目代码库 [14] 中找到。

OSCAR 支持基于容器镜像创建无服务器文件处理服务。这些服务需要用户定义一个脚本，其中包含负责处理的命令。该平台会自动在容器中挂载一个卷，访问调用该服务的文件，执行用户定义的脚本，并加载输出文件夹中的内容。前文所述项目代码库中展示了对图像运行筛选器的基本说明。

IV. 多目标进化算法的开发

SRCNet 由分布于全球的节点网络组成，这些节点相互连接，形成一个完全连接图。然而，资源可用性存在差异，包括偶尔的停机，因此需要定期重新评估节点的连接性和功能执行能力。挑战在于，基于当前的 SRCNet 结构，在特定条件下制定执行计划，以优化执行时间和能耗。这种优化取决于系统规格和各个节点的特征，例如节点间距离、带宽和节点信息。首先，将对系统行为进行理论建模，阐明选择模型函数某些组成部分（例如目标函数及其定义域）以及所施加约束的依据。尽管该解可能无法涵盖整个解空间，但实用指南将有助于

实现更广泛的目标。在实践中，解决问题扩展仍是一个关键考量。目标并非获得最优解，而是推导出一个局部解，使系统请求能够相对高效地执行

A. 问题表述与模型

有必要先明确该问题所要解决的目标，以便最终根据一系列最适合该问题的选择（例如支持算法）来创建正确的软件实现。我们的目标是求向量函数 $F: \Omega \to \mathbb{R}^L$ with $\Omega \subset \mathbb{R}^P$and L, $P \in \mathbb{N}$ 的最小值。起初，人们可能会考虑经典的向量函数优化问题，这需要计算 F 的雅可比矩阵 Jac(F)，即分量 F_i 相对于坐标 x_j 的偏导数矩阵，其中 $i = 1, \ldots, L$ 且 $j = 1, \ldots, P$。我们遇到了第一个问题：这要求 F 可微，但实际情况未必如此。此外，考虑到约束函数，可以使用拉格朗日乘子根据这些约束计算解，但计算成本较高。

我们希望求向量函数各分量的最小值。在此背景下，使用多目标进化算法或许是解决该问题的一种可行方案。因此，我们或许能找到一个次优执行计划；函数 F 越复杂，这种情况就越有可能发生。此外，多目标进化算法在计算效率与实现目标之间能够取得良好平衡，并确保算法不会超出其执行时间。

开发了一种方法，用于模拟单节点系统的行为，并得到符合预期结果的输出。随后，该方法扩展到多节点系统，并在实现时对待最小化的函数进行了泛化。

所开发的模型 F 最初是针对系统中的单个节点设计的，但可推广至 M 个节点和 P 个决策变量（即 F 定义域的维度）。不过，该模型实现时旨在优化执行时间和能耗，这意味着 $L = 2$；但可以看出，尽管这会显著扩展算法，从理论上讲，将函数扩展到 L 个变量并无问题。另一方面，尽管设计时考虑了可扩展性，但该模型实际运行时采用的是 P=3，即 CPU 核心数、GPU 核心数和 ARM 核心数（可以说， P 是指在给定约束条件下，算法能够调整以找到最小值的变量数量，因此，例如，过去几天节点上的负载并未被视为决策变量）。

1) 系统信息索引：SRCNet 将由地理位置分散的节点组成，这些节点共享数据、服务和事件。这将形成一个动态系统，其中节点及上述组件的状态会随时间变化。因此，系统信息索引至关重要，它将作为多目标进化算法引擎的资源。每当系统发生变化时（例如首次启动、节点故障、节点中的某个数据块丢失等），都需要更新算法已知的特征。由于系统会根据发生的变化动态更新，多目标进化算法可能需要重新计算次优执行计划，从而使系统具备一定的动态性。

另一方面，算法正确运行所需的参数化以 JSON 格式的文件表示，其中说明了算法期望特定节点提供信息的方式。我们将该节点标识为 $id = Q$，其参数为 $Q = 1$、...、M，具体如下：

- *ID*：用于标识节点的正整数。在本例中，$ID = Q$。
- *NAME*：节点名称，也可以是其所属国家的名称。
- *CPU*：正整数，表示节点中可用 CPU 核心的数量，与其用途无关。
- *GPU*：正整数，表示节点中可用 GPU 核心的数量，与其用途无关。
- *ARM* 或其他架构：正整数，表示节点中可用*ARM* 核心的数量，与其用途无关。
- 负载：节点过去 x 天的负载。这是一个介于 0 和 1 之间的实数，其中 0 表示节点一直空闲，1 表示节点始终被其他函数占用。
- 连接：节点 Q 与其他节点的连接列表。如果数据块不在该节点上，则会在其直接或间接连接的节点中搜索执行该数据块的位置。此过程将在后文详细说明。
- 数据：每个节点上可用于执行函数的数据块/数据集合列表。

此外，每个节点都具有一组硬件特性，而要提供可靠的近似结果，掌握节点在过去 x 天内的使用情况至关重要。由于数据将在节点之间分布，因此还需要使用一种数据结构，向多目标进化算法提供执行函数所需数据的位置。所有这些信息都存储在系统信息索引中。

2) 解的表示：如前所述，目标函数 F 的定义域位于一个 P 维空间中，值域位于一个 L 维空间中。因此，最优解将是 P 元组 $(x_1, \ldots, x_P) \in \Omega$，满足对于所有 $(y_1, \ldots, y_P) \in \Omega$，$F(x_1, \ldots, x_P) \leq F(y_1, \ldots, y_P)$。

如果推广到 $L \in \mathbb{N}$，则 P 维空间中函数的最小值会在 L 维空间中求得，因为 F 是一个 L 维函数。为了观察可能解的点云，我们最多只能观察到 L=3，因为从计算角度来看，更高维度无法可视化。因此，由于我们希望验证解的行为是否符合预期（这是一个简单的近似，因此二者呈直接关系），图 3 中仅展示了 $L = 2$ 和一个节点的解点云。作为上述内容的示例，下面的解点云表示了在一种更简单但限制性很强的近似条件下，一个节点的行为

（因为假设如果使用 GPU，则不会使用其余部分，其他组件也同样如此）。正如后文将看到的，限制条件会有所放宽，因此预计图中会包含更多的点。

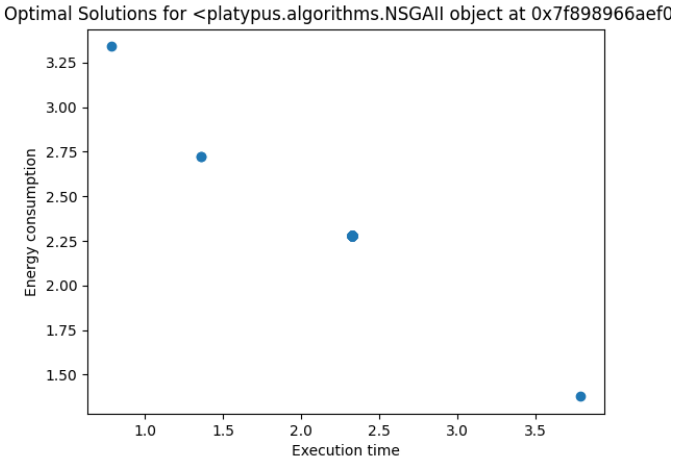

图3. 单个节点的行为。

另一方面，一个解（与前一个点云中的某个点相关联的 P 元组）由以下部分组成（对于单节点系统）：

- CPU 核心：执行期间节点使用的 CPU 核心数量。
- GPU 核心：执行期间节点使用的 GPU 核心数量。
- ARM 核心：执行期间节点使用的 ARM 核心数量。

需要强调的是，解向量中的位置已得到说明。某个决策变量的位置对应于由其 ID 标识的节点的某项特征。稍后在讨论数据访问时，我们将看到顺序的重要性。一般而言，对于包含 M 个节点的系统，情况相同，只是乘以 M。然而，位置十分重要，因为它代表节点的 ID。图4 展示了解的表示方式。

3）目标函数和约束： 到目前为止，这些约束仅用于防止解成为坐标原点。也就是说，系统中没有任何内容被使用，这是不可能发生的。尽管理论上这不可能发生，因为在稠密空间$\mathbb{R}$中，这种情况发生的概率为 0；而且这种情况发生的可能性极低，尤其是随着 P 的值增大，发生概率通常会趋近于 0，但仍将其纳入考虑，因为无论概率多么低，由于计算机的有限表示能力，这种情况仍有可能发生。

对于 P 个决策变量和 M 个节点，所实现的算法如下：

$vars \leftarrow$ 数字列表
过程 NOTVALIDIFONLYZERO
对于 $i \leftarrow 0$ **到** $\text{len}(vars) - 1$ **步执行**
$aux \leftarrow \text{sumarPVariables}(vars)$
如果 $aux = 0$ **则返回** 1。0**结束 if**
结束 for 返回 -1.0**结束过程**

如有需要，可以采用与上述类似的实现方式，将附加约束函数纳入问题。实现 MOEA方法时，我们选择了 Platypus 框架，因为它在处理此类情形时具有很高的灵活性。不过需要注意的是，如果系统当前处于活动状态，则必须重启才能使更改生效。这些约束可表示为 $g_i : \Omega \rightarrow \Lambda$，对于所有 $i = 1, ..., B$，其中 $B \in \mathbb{N}$ 表示约束的数量，且 $\Lambda \subset \mathbb{R}$。如果解 $x \in \Omega$ 满足以下条件，则称其为有效解

$$g_i(x) \leq 0$$

对于所有 $i \in \{1, \ldots, B\}$（注意，Ω 是解空间，也就是 F 的定义域）。在所提出的实现中，$\Lambda = \{-1, 1\}$因为添加的约束决定了 x 是否有效；不存在其他因素，例如解的有效程度。因此，当 $g_i(x) = -1$ 时，x 有效；对于所有 $i = 1, \ldots, B$，若 $g_i(x) = 1$，则其无效。

关于目标函数，目标是同时最小化系统的执行时间和能耗，因此 $L = 2$。由此，我们有 $F : \Omega \rightarrow \mathbb{R}^2$ 定义为$F(x_1, \ldots, x_P) = (F_1(x_1, \ldots, x_P)\ F_2(x_1, \ldots, x_P))$，其中 F_1 表示执行时间，F_2 表示能耗。

在继续之前，需要对目标函数的运作方式作一系列说明。

a) 随机性。由于在这次初步近似中，网络上没有执行数据，因此假设数据服从正态分布。它们不被纳入

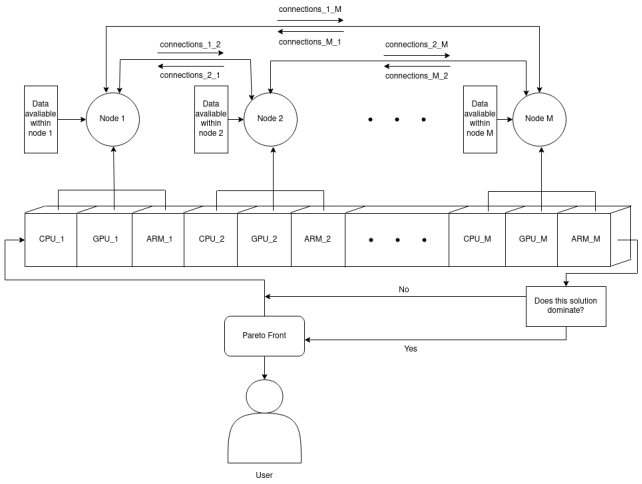

图 4. 示意图展示了算法的工作方式，以及一个解包含哪些组成部分，从而获得执行计划。

决策变量，因为它们与多目标进化算法一样，是外部因素，就像节点负载一样。

b) 访问。为访问一个决策变量，提出以下表达式。如果系统有 M 个节点，每个节点有P 个决策变量，而我想访问 ID 为 N 的节点的第 K 个决策变量，那么如果我们将包含这些变量的数组称为 $vars$，访问表达式为

$$varDecision = vars[P \cdot (N-1) + (K-1)]$$

决策变量的位置以及 $vars$ 的排列顺序都很重要。也就是说，如果节点 N 的 P 个决策变量数组按某种方式排列，那么其他节点也必须遵循相同的顺序，将节点 $N+1$ 的数组附加到右侧，将节点$N-1$的数组附加到左侧。

c) 最短路径。给定节点之间的连接矩阵，如果节点 ii 需要一个自身没有的数据块，它会将函数执行的剩余部分发送到拥有该数据块的节点。首先，获取拥有该数据块的节点列表，然后使用 Dijkstra算法确定其中涉及最短路径的节点。

d)根据核心数量和所用技术体现的并行化行为。作为一种方法，提出函数序列 $\left\{\frac{1}{n^p}\right\}$，其中 n 表示核心数量，p 表示所用技术类型。以 CPU 和 GPU 为例，核心越多，速度越快；而 GPU 比 CPU 更快，因此 $p_{\text{CPU}} < p_{\text{GPU}}$。稍后将针对执行时间说明这一选择的合理性。由于能耗的变化趋势与时间相反，因此提出序列 $\{n^p\}$，理由类似。

假设 $p > 0$ 固定。如果 n_1，n_2 是两个核心数量，且满足 $n_1 < n_2$，那么可以观察到，若 T 是一个核心的执行时间，则有

$$\frac{1}{n_1^p} \cdot T > \frac{1}{n_2^p} \cdot T$$

这符合预期（假设工作均匀或近乎均匀地分配，核心数量越多，预期时间就越短）。这是因为序列 $\left\{\frac{1}{n^p} \cdot T\right\}$ 在渐近意义下趋于 0，也就是说，

$$\lim_{n\to\infty} \frac{1}{n^p} \cdot T = 0$$

此外，核心数量越少，执行时间越长这一论据可由以下内容得出

$$\lim_{n\to 0} \frac{1}{n^p} \cdot T = \infty$$

现在，假设 n 固定，并考虑技术1和技术2，其中技术1比技术2慢，设为 $p_1 < p_2$。若 T 为基准时间，则有 $\frac{1}{n^{p_1}} \cdot T > \frac{1}{n^{p_2}} \cdot T$。此外，值得注意的是，尽管当 $n \to \infty$时，它们渐近趋于0，但收敛速率并不相同。如果我们求解

$$\frac{1}{n^{p_1}} = \frac{1}{n^{p_2}}$$

我们得到 $n^{p_1} - n^{p_2} = 0$，也就是说，我们可以构造

$$n^{p_1} \cdot (1 - n^{p_2 - p_1}) = 0$$

并且由于 $n \neq 0$，因此 $n^{p_2-p_1} = 1$。如果 $p_2 - p_1 \in \mathbb{N}$，根据代数基本定理，我们有 $p_2 - p_1$ 个解。我们只关注解 $n = 1$，因为其余的是复数解或负解。由此可知，如果 $p_1 < p_2$，那么序列 $\left\{\frac{1}{n^{p_1}}\right\}$ 的渐近收敛速度比序列 $\left\{\frac{1}{n^{p_2}}\right\}$ 慢，因此用于区分该技术的近似是准确的。同样可以推出

$$\frac{1}{n^{p_1}} < \frac{1}{n^{p_2}} \text{ if } n \in [0,1]$$

另一方面，需要注意的是，针对某个 n 选定的序列必须根据 p 在其整个定义域上单调递减。当然，这在 $\left\{\frac{1}{n^p}\right\}$ 的情况下成立，其中

$$\frac{d}{dp}\left[\frac{1}{n^p}\right] = -p \cdot n^{-p-1} < 0 \quad \forall p > 0, \forall n > 0$$

接下来，图5展示了根据上述考量，p 取不同值时的图示。

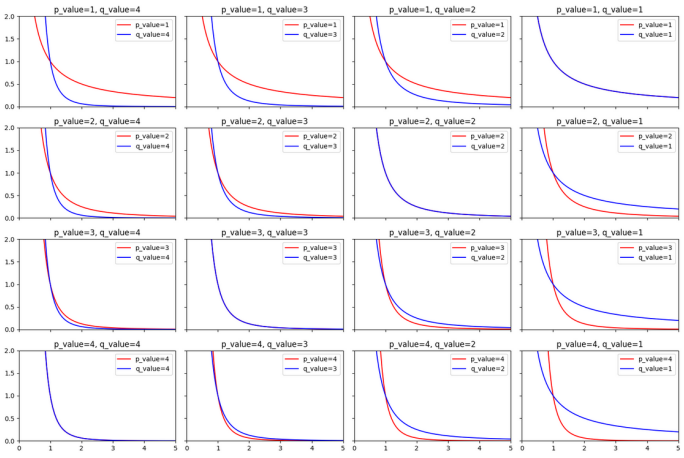

图5. 关于 p的论证。可以看出，对于我们关注的情况（核心数量大于或等于1），p的值越大，图就收敛得越快。

回到目标函数，我们有两个组成部分 $F(x_1, ..., x_P) = (F_1(x_1, ..., x_P)\ F_2(x_1, ..., x_P))$，其中$F_1$ 是执行时间，F_2 是能耗，我们将二者视为相互独立。不过，更现实的近似形式应为 $F_2(x_1, ..., x_P) = (G \circ F_1)(x_1, ..., x_P)$，其中某个标量函数 $G: \Omega \to \mathbb{R}^+$，且 $\Omega \subset \mathbb{R}^+$；或者为 $F_2(x_1, ..., x_P) = G(x_1, ..., x_P, F_1(x_1, ..., x_P))$，其中 G定义为 $G: \Omega \to \mathbb{R}^+$，且 $\Omega \subset \mathbb{R}^{P+1}$。根据我们希望建模的组成部分，对该函数进行详细定义。假设 A 是执行某个函数所需的数据集，# A 是所需数据点的数量，$connectionsMatrix$ 是节点之间的连接矩阵，其中，如果节点i 未连接到 j，则 $connectionsMatrix_{i,j} = -1.0$；如果它们相连，则 $connectionsMatrix_{i,j} > 0$ 和 $i \neq j$；而如果 $i = j$，则 $connectionsMatrix_{i,j} = 0$。此外，尽管最初所有节点彼此相连，但我们也实现了适用于非连通网络的解

（因为可能会出现某些连接失效的情况），在此情境下，我们定义执行时间和能耗。

a)执行时间。按照前文提到的记号，它表示为 F_1，且必须为正值（因为时间不可能为负数，也不可能为零）。给定节点*ID*、执行计划以及执行某项功能所需的数据，对于每个所需的数据块，都会寻找通往包含该数据块的节点的最短路径：如果数据块位于同一节点，则返回 0；如果通过最短路径可以访问到包含该数据块的节点，则返回一个正实数和该节点的 *ID* ；否则，假定该节点与持有这些数据的节点不连通（或者这些数据根本不在网络中），并对其施加惩罚，从而使执行计划无法成立。此方法基于*Dykstra* 的 算法。此外，由于目标是让程序继续在外部节点上执行（将计算带到数据所在处），因此必须考虑在远端节点上的执行时间。假定程序在另一个节点上执行时，不依赖其他节点。也就是说，如果节点 i 发送指令，在节点 j上执行部分功能，则仅提取该节点上需要使用 j 中的数据块的那部分功能的执行时间。如果还需要使用不在 j中的其他数据块，例如位于节点 k上的数据块，则会再次调用该节点。需要注意的是，在建议的节点 i 上执行的时间会单独测量（也就是说，确保每个节点的负载只计入一次）。

另一方面，假设节点中的负载均匀分布，若 $hardware$ 是包含节点相应硬件组件的决策变量数组，则定义$contribution_i$ 为各组件对工作负载的贡献，其值为

$$contribution_i = \frac{hardware_i}{\sum_{j=0}^{j=\text{len(hardware)}-1} hardware_j}$$

若 T 为标准执行时间，则根据 $hardware$ 组件计算的预期时间为

$$\sum_{j=0}^{j=\text{len(hardware)}-1} \frac{1}{contribution_j^{p_j}} \cdot T$$

确保

$$\sum_{j=0}^{j=\text{len(hardware)}-1} contribution_j = 1$$

最后，为考虑节点过去 X 天的负载，在节点 i 上执行某个函数的预期时间为 $time_i = (1 + load_i) \cdot time$，其中 $time$ 是在 $load_i = 0$ 时的预期时间。总之，若节点 i 的负载记为 l_i，函数的处理时间记为 t_{function}，则一般表达式为：若节点 ID 为 i ，且 B 是节点 i 内的数据：

$$F_{1,i}(x_1, ..., x_P) = (1 + l_i) \cdot \left(\sum_{j \in A-B} t_{i,\text{minNode}_j} + t_{\text{minNode}_j,i} + F_{1,\text{minNode}_j}(x_1, ..., x_P) + t_{\text{function}} \right) \quad (1)$$

b)能耗。在前述记法中对应于 F_2，其中很大一部分采用与执行时间类似的计算方法，依靠 *Dykstra* 估算其他节点上的最短执行时间，这些节点存放着不在同一节点上的数据块；然后结合可用硬件组件推算能耗。在这种情况下，尽管分布也是均匀的，但计算方法有所不同。也就是说，若 $hardware$ 为硬件组件，即问题的部分决策变量，并与节点 i 相关联，则针对 $j = 0, ...,$len(hardware)-1 得到 $contribution_j$，并计算节点的预估能耗如下：

$$\sum_{j=0}^{j=\text{len(hardware)}-1} contribution_j^{p_j} \cdot T$$

能耗的一般表达式与时间的表达式类似，但需作出上述调整。对于某个节点 i，若将 B 记为节点内的数据，将 E_{function} 记为处理函数时的能耗，则表达式为：

$$F_{2,i}(x_1, ..., x_P) = (1 + l_i) \cdot \left(\sum_{j \in A-B} E_{i,\text{minNode}_j} + E_{\text{minNode}_j,i} + F_{2,\text{minNode}_j}(x_1, ..., x_P) + E_{\text{function}} \right) \quad (2)$$

B. 实现

本实现使用 Python 完成，并借助 Platypus 框架实现多目标进化算法（MOEA）。具体而言，我们采用了 NSGA-II 算法，种群规模为 100，并使用 RandomGenerator 生成初始种群。解的选择采用二元锦标赛选择，而模拟二进制交叉（SBX）则用作重组算子。初步测试在 espSRC 计算基础设施提供的虚拟机（VM）上进行 [15]，该虚拟机还部署了支持函数即服务（ FaaS）功能所需的 Kubernetes 平台。

V. 结论

在这项工作中，我们希望强调设计方案的重要性：该方案应有效应对将计算直接移至数据所在地这一挑战，从而克服移动海量数据所带来的困难。采用 FaaS 模型是一个重要选项，它支持在分布式基础设施中远程执行函数。我们提出了一项初步

方案，旨在 SRCNet 的背景与架构下，采用基于多目标进化算法的方法优化 FaaS 上流水线的执行计划。我们的方法旨在促进数据网格中的决策，针对执行时间和能耗等关键方面优化数据上的计算操作。该方法可集成到代理器中，从而更高效、灵活地管理 SRCNet 架构中的分布式计算资源，并推动大规模天文数据处理朝着更高效的方向发展。

作为未来工作，我们建议扩展所设计的算法，纳入基于当前负载的实际指标和节点负载数据，而非像迄今为止那样依赖仿真或理论模型。这将有助于做出更精准、更具适应性的决策，进一步提升系统在动态运行环境中的效率和响应能力。未来研究的另一个重要方向是详细研究函数和数据操作的能耗，从而更有效地为多目标进化算法提供数据，并提升其资源优化性能。最后，还需全面评估采用多目标进化算法的代理器的集成情况和有效性，以衡量本文所提方法的效能和实际影响。

致谢

我们感谢由 MCIU/AEI/10.13039/501100011033 资助的项目 CEX2021-001131-S，以及由安达卢西亚自治区政府经济转型、工业、知识和大学部资助的项目 DOC01497，并感谢安达卢西亚 ESF 2014-2020 运营计划的支持。我们感谢由 MCIU/AEI/10.13039/501100011033 资助的项目 TED2021-130231B-I00，以及“欧盟 NextGenerationEU/PRTR”的支持。我们还感谢由 MCIU/AEI/10.13039/501100011033 资助的项目 PID2021-123930OB-C21，以及“欧洲区域发展基金‘建设欧洲之道’”的支持。

参考文献

[1] J. Garrido、L. Verdes-Montenegro、S. S´anchez 和 J. Gallardo，《平方公里阵列项目与平方公里阵列区域中心网络的现状》，《西班牙天体物理学进展 *XI*》，第 360 页，2023 年。[2] P. Hartley、A. Bonaldi、R. Braun、J. Aditya、S. Aicardi、L. Alegre、A. Chakraborty、X. Chen、S. Choudhuri、A. Clarke 等，《平方公里阵列科学数据挑战赛2：分析与结果》，《皇家天文学会月刊》，第 523 卷，第 2 期，第 1967–1993 页，2023 年。[3] A. Bonaldi、T. An、M. Br¨uggen、S. Burkutean、B. Coelho、H. Goodarzi、P. Hartley、P. Sandhu、C. Wu、L. Yu 等，《平方公里阵列科学数据挑战赛1：分析与结果》，《皇家天文学会月刊》，第 500 卷，第 3 期，第 3821–3837 页，2021 年。[4] I. A. Machado、C. Costa 和 M. Y. Santos，《数据网格：数据架构范式转变的概念与原则》，*Procedia Computer Science*，第 196 卷，第 263–271 页，2022 年。[5] A. P. Rajan，《云计算中无服务器架构——函数即服务（FaaS）综述》，*Telkomnika*（电信、计算、电子与控制），第 18 卷，第 1 期，第 530–537 页，2020 年。

[6] E. Jonas、J. Schleier-Smith、V. Sreekanti、C.-C. Tsai、A. Khandelwal、Q. Pu、V. Shankar、J. Carreira、K. Krauth、N. Yadwadkar 等，《简化云编程：伯克利对无服务器计算的观点》，*arXiv* 预印本 *arXiv:1902.03383*，2019 年。[7] A.-C. Orgerie、M. D. d. Assuncao 和 L. Lefevre，《提升大规模分布式系统能源效率的技术综述》，*ACM Computing Surveys (CSUR)*，第 46 卷，31，2014 4，1第。–。[8] H. Hussain、S. U. R. Malik、A. Hameed、S. U. Khan、G. Bickler、N. Min-Allah、M. B. Qureshi、L. Zhang、W. Yongji、N. Ghani 等，《高性能分布式计算系统资源分配综述》，*Parallel Computing*，第 39 卷，第 11 期，第 709–736 页，2013 年。[9] N. Yi、J. Xu、L. Yan 和 L. Huang，《基于改进蚁群算法的分布式信息物理系统任务优化与调度》，*Future Generation Computer Systems*，第 109 卷，第 134–148 页，2020 年。[10] D. Habenicht、K. Kreutz、S. Becker、J. Bader、L. Thamsen 和 O. Kao，《Syncmesh：提升网状边缘网络中函数即服务的数据局部性》，载于第五届边缘系统、分析与联网国际研讨会论文集，2022 年，第 55–60 页。[11] M. Chadha、A. Jindal 和 M. Gerndt，《计算密集型函数即服务函数的架构特定性能优化》，载于*2021 IEEE* 第 *14* 届国际云计算会议（*CLOUD*）。IEEE，2021 年，第 478–483 页。[12] B. Ahmadi、O. Ceylan、A. Ozdemir 和 M. Fotuhi-Firuzabad，《分布式能源规划与存储管理的多目标框架》，*Applied Energy*，第 314 卷，文章编号 118887，2022 年。[13] W. Ma、R. Wang、Y. Gu、Q. Meng、H. Huang、S. Deng 和 Y. Wu，《基于知识驱动进化算法的多目标微服务部署优化》，*Complex & Intelligent Systems*，第 7 卷，第 1153–1171 页，2021 年。[14] A. Rodriguez 和 M. Parra-Roy´ón，《用于 SRC经纪的多目标进化算法》，2024 年 1 月。[在线]。网址：https://github.com/AlvaroRodriguezGallardo/src-brokering/ [15] J. Garrido、L. Darriba、S. S´ánchez-Exp´ósito、M. Parra-Roy´ón、J. Mold´ón、M. ´A. Mendoza、S. Luna-Valero、A. Alberdi、I. M´árquez 和 L. Verdes-Montenegro，《迈向全面参与开放科学的西班牙平方公里阵列区域中心》，*Journal of Astronomical Telescopes, Instruments, and Systems*，第 8 卷，第 1 期，2021 年。