Kubernetes 资源配置推荐系统设计说明书

杜雷鸣

一、背景

Kubernetes 提供了对 CPU、内存等 IT 资源的抽象能力，用户可以根据容器的实际需求声明资源规格，这种方式有效提升了集群资源的管理效率。然而，如何填写容器的资源规格一直以来都是困扰开发者和资源管理员的关键问题，过高的资源规格会导致大量的资源浪费，而过低的规格又会为应用带来潜在的稳定性风险。

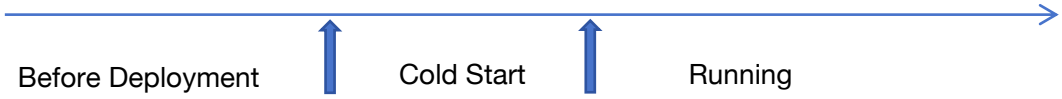
根据 Sysdig 的报告，业界的 kubernetes 集群中，有 69%的 CPU 和 18%的内存处于已分配但尚未使用的状态，也符合常见的开发者通常会调高资源需求的行为预期，此类问题通常会导致大量的 IT 资源浪费，提高 IT 资源的成本。伴随着 AI 在工业和商业界的大量引入以及大模型带来的 AI 相关的计算资源成本的快速升高，IT 资源的浪费已逐步成为一个不可忽视的问题。

本文旨在设计一个高效率、高准确率的容器资源推荐系统，既能高效率的帮助开发者和资源管理者配置合适的容器资源规格，又能更好的辅助业务，降低 IT 成本，从而达到降本增效的目的。

二、资源配置推荐系统需求分析

构建一个系统的首要核心问题是较为全面合理的用户场景和需求分析。

根据业务开发和部署的全生命周期以及系统所面向的客户和具体的客户场景，我们将需要资源动态配置的生命周期进行划分如下：



主要包含 Before Deployment, Cold Start 和 Running 三个阶段。

1. Before Deployment 阶段

这一阶段主要面向的场景是开发者在代码开发完成后进行部署的阶段，此阶段业务尚未开始部署运行，需要在代码部署前给予开发者相应合理的资源推荐。

2. Cold Start（冷启动）阶段

冷启动阶段主要面向的场景是开发者在完成部署后，承载业务的 POD 尚未收集到足量的历史运行数据并且业务可能尚未进入稳定运行的阶段，此阶段既是业务的冷启动阶段也是 IT 资源消耗的冷启动阶段（通常在业务部署前期 IT 资源可能消耗较大）。

3. Running 阶段

长期持续运行阶段面向的场景是经过部署，业务各项运行参数已经经过了较长时间的迭代，进入了业务稳定运行期，需要在业务稳定运行期给予合理的资源推荐。

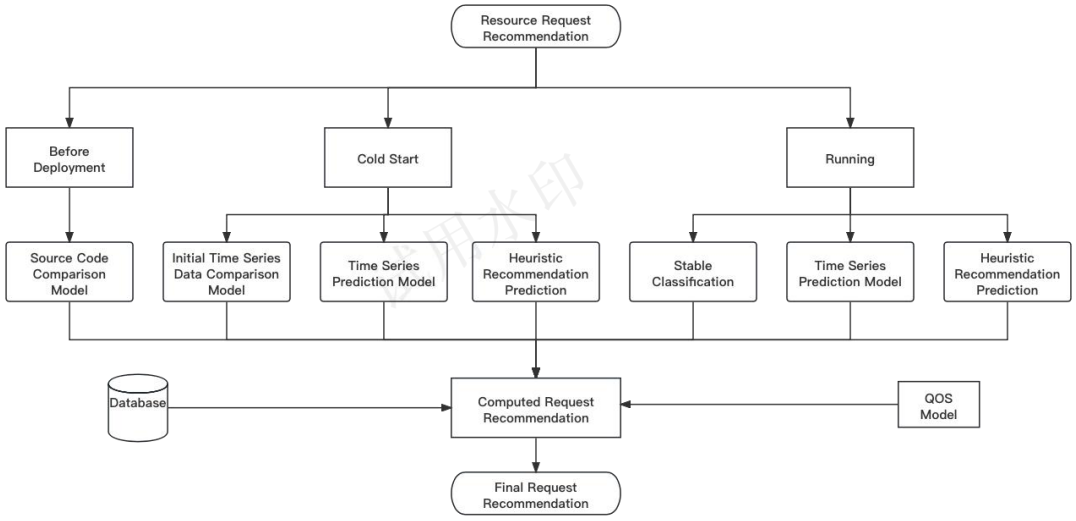
三、数据分析与整体方案

3.1 数据分析

在对业务场景进行初步区分和总结后，**首先应就业务具体情况进行大量数据分析**，数据分析内容包括部署前的业务类型和代码类型以及它们的分布等，部署后的不同阶段的时间序列数据分析和整体业务分布的数据分析等。（由于缺少业务数据，此处略过具体分析)

3.2 整体方案

通过数据分析和业务规划的整体方案如下：



本系统的输入为需要进行资源推荐的业务代码或者已经部署上线的对应 pod 的 CPU 和 Memory 的历史监控值，输出为对应 pod 的 CPU 和 Memory 的 request 值。

资源推荐问题的本质是预测，是为业务在未来一段时间内的运行情况进行资源推荐。本系统整体的核心运作思想如下：

1. 针对具体的业务，如果历史记录中有相似的业务，直接查询得到对应的资源推荐值。
2. 无法直接查询的业务，基于不同类型序列的时间序列预测，根据预测的数据进行推荐值计算。
3. 对于无法预测的业务，使用启发式算法根据历史值进行合理资源推荐。

3.3 推荐值计算方案

由于实际业务对 CPU 和 Memory 的容忍度不同，因此我们采用不同的推荐值计算模型对 CPU 和内存推荐值进行计算。

对于 CPU 推荐值的计算，考虑到在 CPU 资源少量不足时，可能对业务 QOS 有所影响，因此我们采用基于 QOS 的折算系数的乘法模型，如下式所示。

$$CPU_{recomm} = CPU_{compute} * factor_{QOS}$$

针对 Memory 类型，考虑到在内存资源不足时，业务极易导致 OOM，对业务产生较大干扰，因此对于 Memory 的推荐值计算，我们采用基于 QOS 折算系数的加法模型，如下式所示。

$$MEM_{recomm} = MEM_{compute} + factor_{QOS}$$

按照本文的系统设计，需要根据业务类型、运行数据等实际数据构建对应的 QOS 模型，来决定对应的$factor_{QOS}$。由于缺少实际业务数据，我们在 Demo 中不做具体实现，两个$factor_{QOS}$均按照固定值来计算，在这里 CPU 的$factor_{QOS}$设置为 1.1 倍，Memory 的$factor_{QOS}$设置为 100MB。

在 compute 值的计算方法上，针对不同类型的数据情况，本系统采取三种资源推荐值的计算方法:

1. 针对需要查询数据库和历史对比的业务，我们取所查询到的历史资源推荐值直接作为相应的推荐值。
2. 针对基于预测数据的计算值选择，我们取历史数据加上预测数据 CPU 的 95 分位数值和 MEM 的最大值作为计算值，并将计算值与对应$factor_{QOS}$计算从而得到最终的推荐值。
3. 针对无法预测且没有历史对比的业务，我们采用加权滑动窗口的方式得到对应的计算值，并将计算值与对应$factor_{QOS}$计算得到最终的推荐值。

四、详细设计方案

4.1 Before Deployment 阶段

Before Deployment 阶段的典型用户场景是，用户在编写完代码进行打包上传准备开始部署具体业务时需要进行资源规格选择。因而此处需要一个可以根据源代码评估资源规格的模型。

解决这个问题的核心是构建一个可以根据源代码的特征预测未来可能的资源需求的模型，本系统提供以下两类解决方案：

1. 与数据库中代码特征比对直接得到资源推荐值

使用启发式算法进行代码部分特征的比对，从而获取对应的相似度较高的代码记录，并将对应的资源用量作为推荐值。常见的算法包括 TF-IDF, Simhash，SSDeep,LSA 等基于文本相似度检测的无监督算法，AST 等基于结构语法树的相似度检测算法，基于神经网络的 DSSM 等算法，基于 Graph 模型的代码相似度检测算法，基于大模型的代码向量抽取算法以及其他基于监督模型的特征比对算法等。

由于缺少语料库，此处我们在 Demo 中仅实现基于 Simhash 的文本相似度比对算法，更多基于结构或监督学习或大模型的算法需要根据具体数据来评估准确性。

SimHash 算法是 Google 在 2007 年发表的论文《Detecting Near-Duplicates for Web Crawling》中提到的一种指纹生成算法，被应用在 Google 搜索引擎网页去重的工作之中。简单的说，SimHash 算法主要的工作就是将文本进行降维，生成一个 SimHash 值，也就是论文中所提及的“指纹”，通过对不同文本的 SimHash 值进而比较汉明距离，从而判断两个文本的相似度。

在 Demo 中，我们将汉明距离的阈值设置为 30，即在数据库中筛选第一个汉明距离小于 30 的记录的资源值作为当前业务代码的资源推荐值。

2. 构建源代码和源代码特征到运行资源需求的监督模型（Demo 中不做实现）

构建一个源代码到运行时资源需求的监督模型，输入为对应的源代码或源代码的特征向量，通过代码的语义信息，结构信息，业务类型等特征，输出预测未来的资源需求。该方案的好处在于在精确度能保证的前提下，可以对各种不同的业务代码进行全面覆盖。方法选择上可以选择使用 XGBoost，GBDT 或基于神经网络的回归算法等进行监督模型的构建。

4.2 Cold Start 阶段

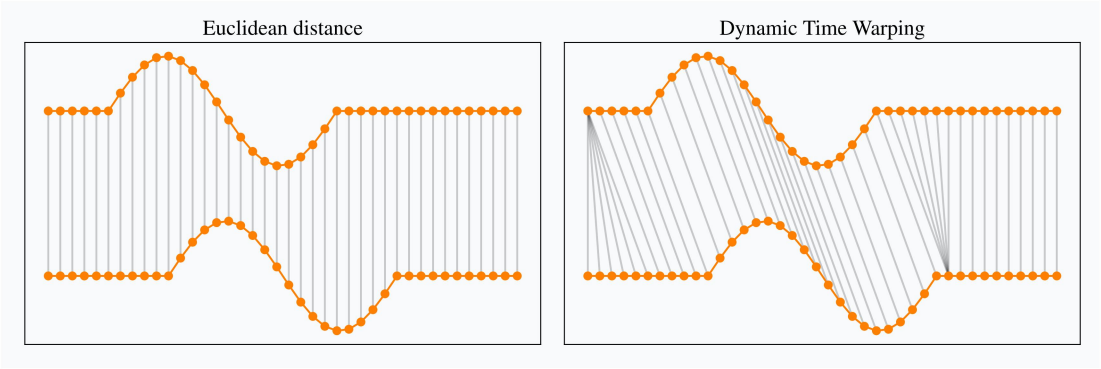
Cold Start 阶段是业务在运行起来以后的初级阶段，在此阶段对应的 pod 上尚未累积大量的运行数据，并且对应的业务和 IT 资源可能有比较大的波动。因此需要根据少量数据来进行资源配置推荐。

本系统提供以下两类解决方案：

1. 对比查询已有数据库中的类似样本进行资源推荐

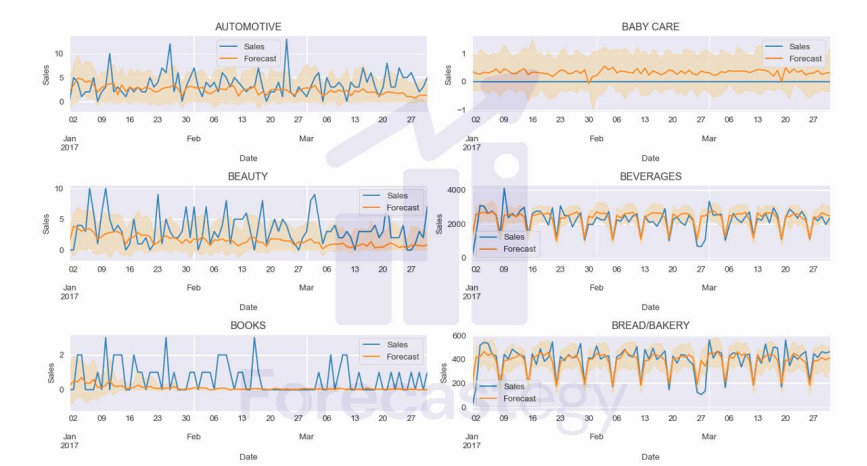
我们将待推荐业务在冷启动阶段的部分数据与数据库中已有历史记录进行对比，相似度高的可以直接取对应的资源推荐值作为最终的推荐值。**此类问题的本质是时间序列相似度的对比**，常见的算法有 DTW 算法、欧氏距离、互相关函数、cosine 相似度以及其他监督学习算法。此处我们采用 DTW 算法作为时间序列相似度的计算算法。

动态时间规整（Dynamic Time Warping，DTW）是按照距离最近原则，构建两个长度不同的序列元素的对应关系，评估两个序列的相似性。在构建两个序列元素对应关系时，需要对序列进行延伸或压缩。



2. 冷启动阶段时间序列预测，根据预测数据计算推荐值

如果无法找到相似的运行数据，则采用适合冷启动阶段的时间序列预测模型进行预测，根据预测的数据采用启发式算法计算对应的资源推荐值。常见的冷启动时间序列预测算法有基于 LSTM 的多维度时间序列预测算法(如 Amazon 的 DeepAR)或基于机器学习的回归算法等。此类算法需要实际数据进行模型训练，Demo 中我们不做实现。

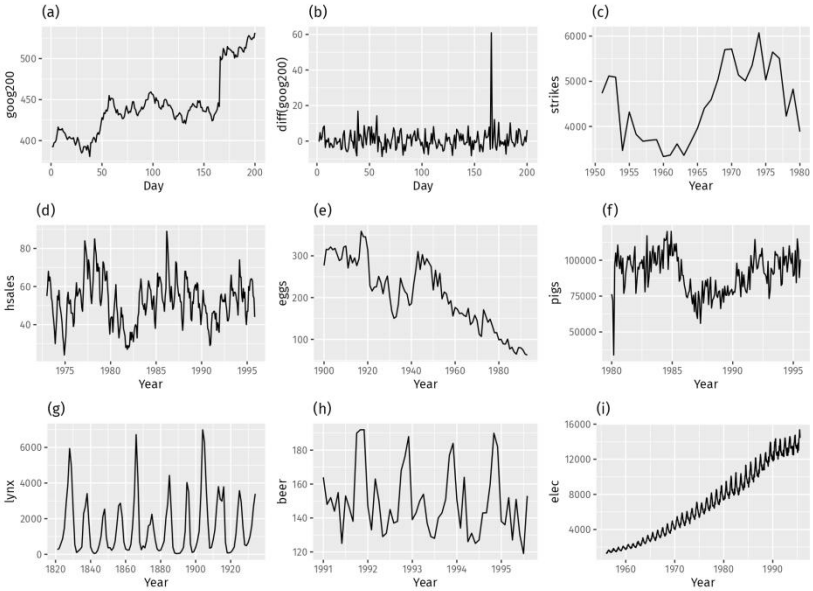


3. 启发式算法计算资源推荐值

当以上两种算法都无法进行资源推荐时，我们采用启发式算法方案，对冷启动阶段的历史数据基于滑动窗口进行加权平滑，根据加权平滑后的数据进行资源推荐计算。对这一阶段的 CPU 推荐，我们取 99 分位数作为计算值，对 Memory 推荐，取最大值作为计算值。

4.3 Running 阶段

在持续运行阶段，由于已经积累了大量的历史运行数据，可以根据已有的时间序列数据进行预测，根据预测值计算资源推荐值。如图所示，实际运行的资源变化情况可能有多种情况，不同时间序列模型的预测效果也不尽相同，因此我们针对不同业务情况采取以下三种不同的方案。



1. 业务相对平稳，波动较小的时间序列

此类时间序列无需进行时间序列预测，可以根据历史值决定资源推荐值。常见的时间序列平稳分类算法包括阈值检测、聚类、监督学习模型等算法。在 Demo 中，我们开发了基于 CV 系数（变异系数）的阈值检测和基于 CV 系数的聚类算法（KMeans）两种算法。

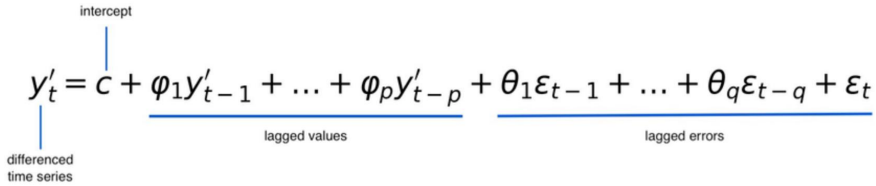
此处的平稳可以理解为资源在运行过程中没有波动，但在实际的业务中，资源的波动是不可避免的，并且可能会存在方向一致的 bias，因此，我们首先用阈值来进行较为严格的

平稳性判别，阈值无法判别的，我们采用聚类的方式来适配业务中可能固有的波动。我们采取滑窗+CV 系数的方式来构造特征，滑窗可以在不同维度反映序列波动特征，而 CV 系数的计算则在波动的时候考虑到了 base 的影响。

2. 适合系统中预设的多种时间序列预测模型的序列

针对不同的时间序列预测，业界有不同的准确率较高的算法。如基于传统参数的 AR,MA,ARMA, ARIMA,SSM 等，基于序列分解的算法，基于机器学习的回归预测算法和基于深度学习的 LSTM 等的时间序列预测算法。此处我们提供多种不同类型的时间序列预测算法以应对不同的数据类型。在实际使用时，优先进行模型准确率校验后再进行时间序列预测，根据预测的资源序列数据和历史数据共同决定资源推荐值。在 Demo 中我们引入了常用的 ARIMA 和 Holt Winters 算法进行一维时间序列预测。

ARIMA:



Holt Winters algorithm:

$$
\begin{aligned}
\text{level} \quad L_t &= \alpha(y_t - S_{t-s}) + (1-\alpha)(L_{t-1} + b_{t-1}); \\
\text{trend} \quad b_t &= \beta(L_t - L_{t-1}) + (1-\beta) b_{t-1}, \\
\text{seasonal} \quad S_t &= \gamma(y_t - L_t) + (1-\gamma) S_{t-s} \\
\text{forecast} \quad F_{t+k} &= L_t + k b_t + S_{t+k-s},
\end{aligned}
$$

在采用时间序列模型预测的过程中，本系统的逻辑是，优先使用历史数据进行模型的准确性校验，在准确性满足要求的情况下再进行时间序列预测，此模块后面可以扩展为基于机器学习的模型预测或多模型混合预测算法。

3. 除以上两种类型以外的其他时间序列类型

由于波动较大或预测准确率较低，可以根据历史数据基于滑动窗口的启发式算法对未来资源推荐值进行合理预测。

4.4 QOS 模型

进行资源推荐的本质是在保证业务稳定运行的前提下尽量减少设置资源与实际运行资源需求的差异，保证业务稳定运行。业务稳定运行的指标在云计算中称为 QOS（Quality of Service），通常包含 bandwidth, latency, number of OOM, QPS 等。不同业务对于 QOS 和具体 QOS 的指标依赖不同，最终反映到对不同 CPU 和 Memory 配置的容错度不同，因此需要一个 QOS 模型为不同的业务设置对应业务的安全阈值空间，以满足业务 QOS 的需求。

针对 QOS 模型，可以通过将业务类型，时间序列运行数据，运行特征，代码特征等业务和开发特征作为输入，对应的$factor_{QOS}$作为输出构建一个监督模型，以此满足最终业务的 QOS 需求。由于构建监督模型需要业务数据，因此在 Demo 中，我们固定将 CPU 的$factor_{QOS}$设置为 1.1 倍，MEM 的$factor_{QOS}$设置为 100MB。

五、结论与展望

本文设计了一个基于业务代码开发和部署流程的全生命周期 CPU 和 Memory 运行资源配置推荐系统，基于较为全面的需求分析，为业务部署的不同生命周期和不同部署阶段提供了较为完备合理的资源配置推荐，并提供了多种数据驱动的闭环算法来应对不同类型的业务和资源变化需求。

5.1 是否有其他解决方案？

本系统提供的是基于业务数据分析和数据驱动的较为具体的推荐方案。同时我们可以采取基于序列决策过程的端到端资源推荐方案，如基于强化学习的资源需求推荐方案，基于贝叶斯的资源需求推荐方案等。此类方案的好处是在模型稳定后，单步的资源决策可以更优，但缺点在于需要大量的数据和实验进行模型收敛，且容易收敛不稳定。

5.2 资源推荐系统是否有其他用处？

本系统的本质是通过引入 AI 的方式对业务开发和部署的不同阶段进行智能动态画像，就系统而言，其画像结果不仅仅可以用作资源推荐，也可以作为 Kubernetes 集群一次和二次调度的输入，可以为资源调度和管理提供更好的建议，降本增效。同时也作为资源和业务分布的数据分析来源，为当下和未来的 IT 资源更新和业务更新提供进一步数据驱动的建议。

5.3 资源推荐系统所处位置的思考

从业务代码优化和 IT 资源管理的现状来看，代码的开发时和运行时是相对比较割裂的。从业界的工作上来看，当下 AI 引入代码开发也大多数是辅助替换传统的一些比较耗时的工作或者集中于解决开发过程的效率问题，IT 资源管理本身也仅仅是引入 AI 来辅助更好的资源管理或降本增效。从本系统的实践来看，基于业务的代码开发时和运行时不应该是割裂的。从全局的角度看，引入 AI 到 IT 领域时应该将开发时和运行时作为一个统一的整体，引入更多的工具促进代码开发时和运行时的相互优化，即在开发过程中引入代码运行资源优化，在资源优化过程中引入开发逻辑优化，将会高效促进开发时和运行时的“软硬件联合优化”。