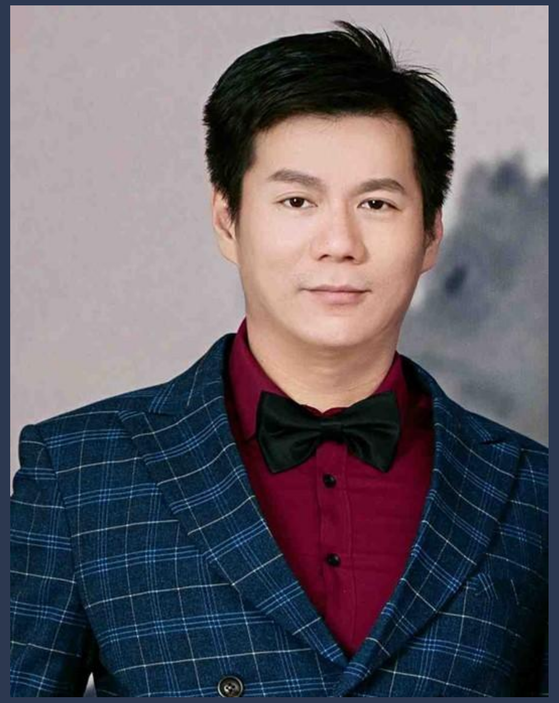
基于时间序列数据预测模型的智能量化交易系统性能优化实践

黄益聪

楷同科技
AI Empowers Trading



黄益聪
楷同科技 CEO

曾担任 Intel 高级工程师，阿里巴巴高级技术专家，中国互联网百强独角兽企业技术副总裁、CTO
长期从事大数据与人工智能领域研究，有15项中国发明专利，3项美国发明专利。目前负责AI 量化交易系统实践。

Agenda

- 智能量化交易系统概况
- 全链路数据流优化
- 服务化AI模型介绍
- 总结与展望

量化交易

量化交易是利用计算机程序和数学模型来制定和执行交易策略的一种交易方式。它基于对市场数据进行分析和模式识别，通过确定性规则或者数理模型来进行决策，而非依赖于主观判断或情绪

行业概况-量化交易发展历程

20世纪70年代：**量化交易初期**

20世纪70年代，量化交易开始萌芽。当时的交易员利用计算机对市场数据进行分析，从中寻找投资机会。这些交易员通常是在机构投资公司工作，他们的计算机分析基于传统的统计学方法。

20世纪80年代：**复杂数学模型**

20世纪80年代，股票期权成为主流的金融工具之一。期权交易需要高效的定价模型，因此，量化交易开始应用更复杂的数学模型，如随机漫步模型和布朗运动模型。

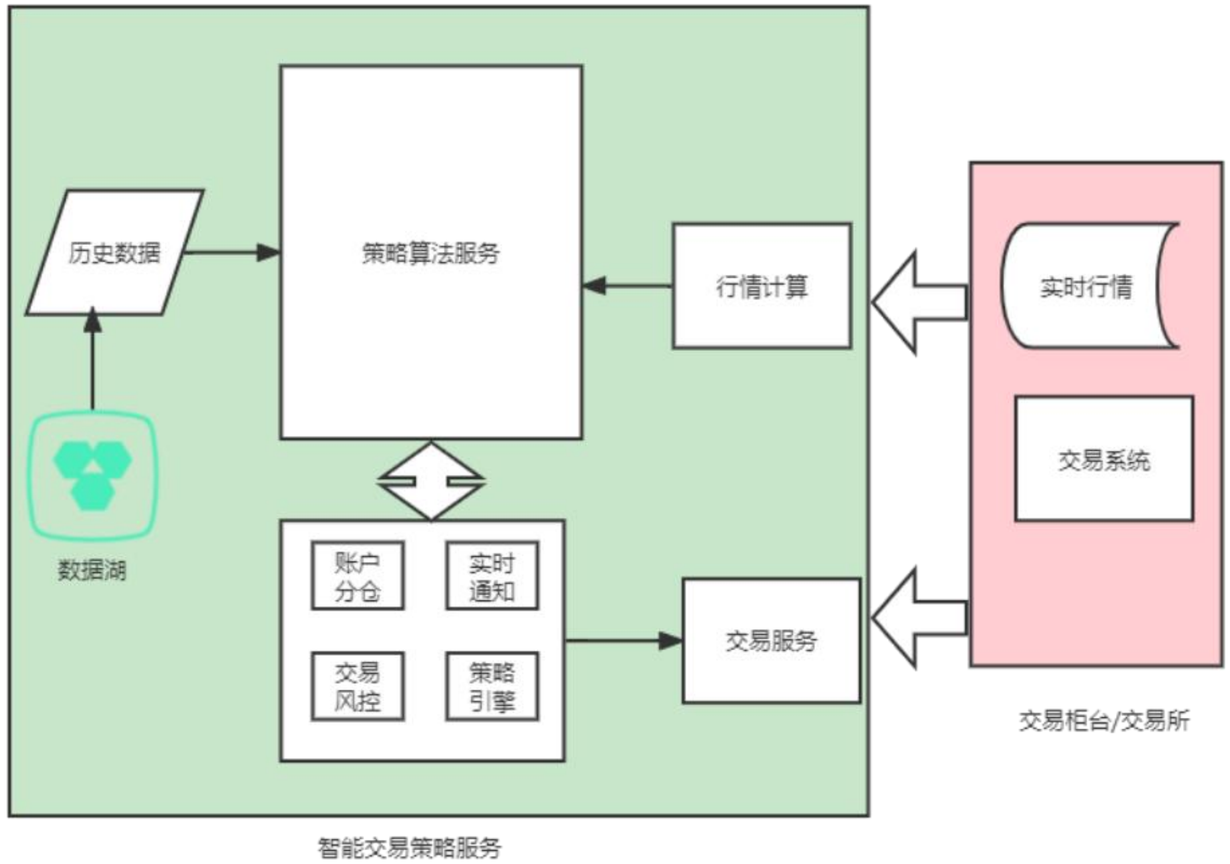
20世纪90年代：**高频交易和算法交易**

这些交易使用计算机算法进行交易决策，并以超快速度进行交易。这种交易方式需要高度优化的算法和技术，这使得量化交易得到了进一步的发展。

21世纪初：**机器学习和人工智能的应用**

随着计算机技术和数据分析技术的进步，量化交易开始应用更复杂的数学模型，如机器学习和人工智能。这些技术使得交易系统能够更准确地预测市场趋势和价格变动。

AI量化交易系统



海量数据分析

自动收集、清洗、存储、处理和分析大量的历史市场数据，包括价格、指数、财务数据等。通过分析数据来发现交易机会。

风险控制

通过各种手段来控制风险，包括资产配置、仓位控制、止损、止盈、预警、自动熔断等

智能算法设计

使用各种复杂算法来分析市场数据和寻找交易机会。产品包括多个可复用、自由组合的最先进算法模型，能够优化交易策略并获得更高的收益。

极速执行

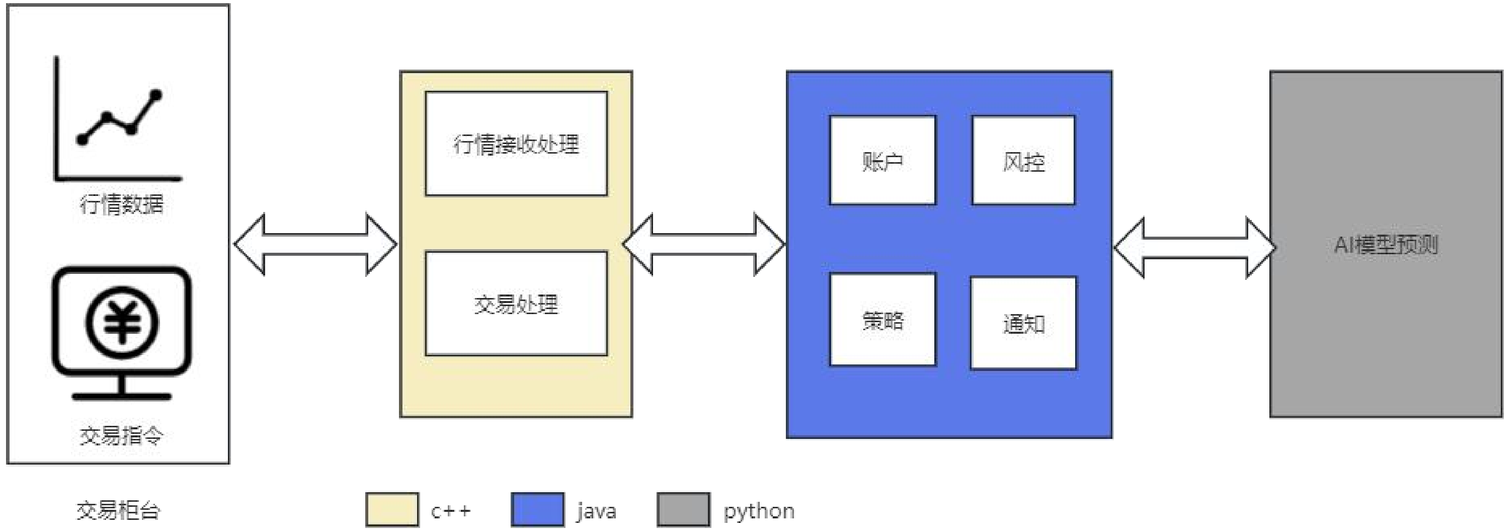
快速地接收和处理市场数据，并通过高效的交易技术将交易指令发送到市场中

数据化运营管理

提供易用的管理后台进行数据管理、算法优化、技术支持等

实时数据流

- Tick级行情数据
- 全链路秒级处理时延
- 多语言开发系统交互



多语言开发的量化交易系统

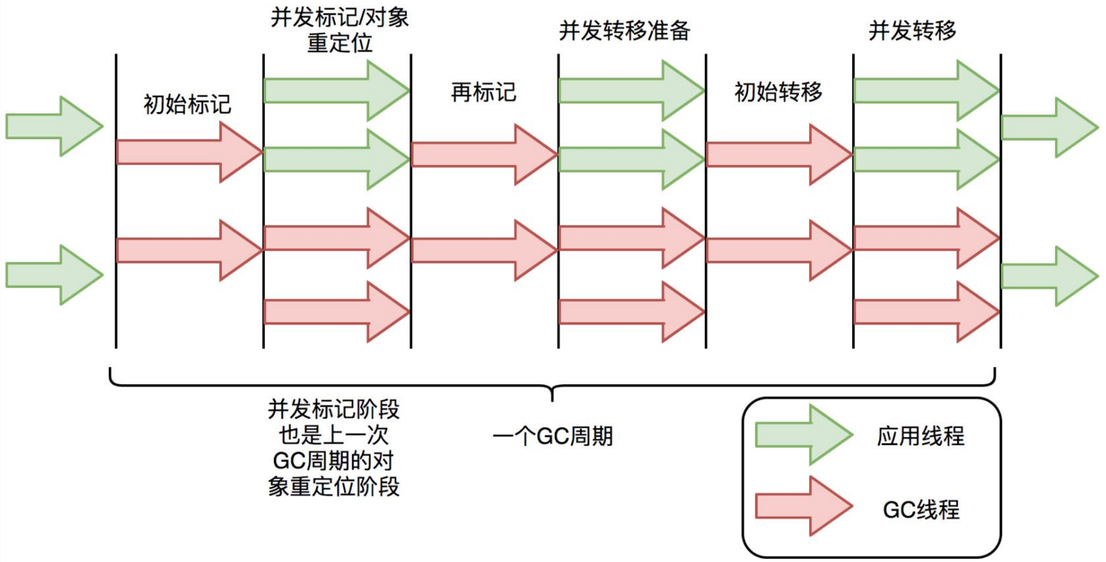
实时数据流性能挑战

- 数据高频产生：tick数据
- 数据量大：全市场数万品种
- 数据生命周期长：时间序列数据预测，t时刻产生的数据，生命期至t+n
- 多语言开发的系统交互：c++, java, python

性能挑战：秒级处理时延，内存gc

数据流性能优化尝试

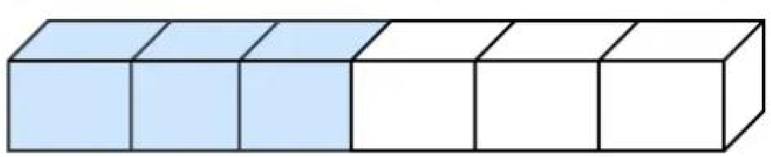
- 业务层面：数据采样降频
- 系统层：Linux HugePage
- 数据交互：Direct Buffer
- Jdk应用层：Tune java GC: CMS GC、G1 GC、



Zgc减小gc延迟

数据流内存优化解决方案

- Jdk unsafe方法直接管理内存
- 循环共享内存池存储行情数据



数据循环写入预分配内存池

优化效果：2g java heap, TP99<2ms

```
try{
    us = getUnSafe();
    bufferSize = getBuffSize(num_step, price_level);
    buffer = us.allocateMemory(bufferSize);
    buffer_ready = true;
}
```

java申请共享循环内存池

```
addr_int = (int *)((long)record_start + sample_offset);
*addr_int = p->UpdateMillisec;
sample_offset += intSize;

addr_double = (double *)((long)record_start + sample_offset);
*addr_double = p->OpenPrice;
sample_offset += doubleSize;

addr_double = (double *)((long)record_start + sample_offset);
*addr_double = p->LastPrice;
sample_offset += doubleSize;

addr_double = (double *)((long)record_start + sample_offset);
*addr_double = p->HighestPrice; //price should >0
sample_offset += doubleSize;

addr_double = (double *)((long)record_start + sample_offset);
*addr_double = p->LowestPrice;//price should >0
sample_offset += doubleSize;

addr_int = (int *)((long)record_start + sample_offset);
*addr_int = p->Volume;
last_volume = p->Volume;
sample_offset += intSize;
```

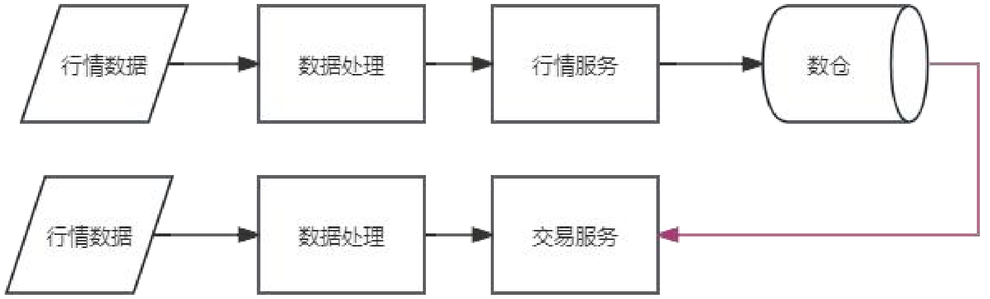
C++写入行情数据

```
s.open = (float)us.getDouble(buf);
buf += doubleSize;

s.close = (float)us.getDouble(buf);
buf += doubleSize;

s.high = (float)us.getDouble(buf);
buf += doubleSize;

s.low = (float)us.getDouble(buf);
buf += doubleSize;

s.volume = (float)us.getInt(buf);
buf += intSize;

s.amount = us.getDouble(buf);
buf += doubleSize;
```

java读取行情数据

数据流快速恢复

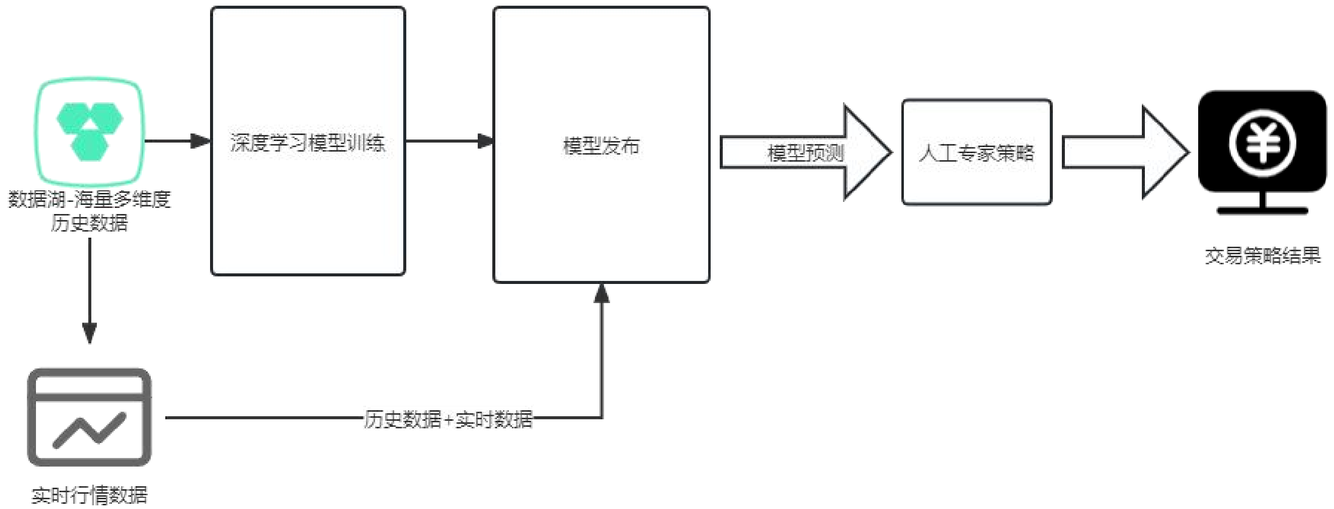
- 拆分行情服务、交易服务
- 行情数据异步存储实时数仓
- 守护进程监控服务健康状态
- 交易服务重启，快速读取断点之前t–n,…t的数据



AI模型预测服务

需求

- 高性能 TP99<30ms
- 版本部署热更新
- 模型快速回滚
- 多框架支持，tf and torch



AI预测服务性能优化

- Use AVX2, FMA
- 数据精度裁剪
- java client to server
- Grpc prior to http
- 重用对象, 重用连接
- Batch predict
- Warm-up

尝试：使用LLM转换torch to tf

```python
def forward(self, x):

    if self.flag:
        x = x.permute(0, 2, 1, 3)
    else:
        x = x.permute(0, 3, 1, 2)  # N, W, C, H
    N, W, C, H = x.shape
    x = x.contiguous().view(N * W, C, H)

    # Transformations
    x = self.to_qkv(x)

    qkv = self.bn_qkv(x)
    q, k, v = torch.split(qkv.reshape(N * W, self.heads, self.dim_head_v * 2, H),
                          [self.dim_head_v // 2, self.dim_head_v // 2, self.dim_head_v], dim=2)

    # Calculate position embedding
    all_embeddings = torch.index_select(self.relative, 1, self.flatten_index).view(self.dim_head_v * 2, self.dim, self.dim)
    q_embedding, k_embedding, v_embedding = torch.split(all_embeddings, [self.dim_head_qk, self.dim_head_qk, self.dim_head_v], dim=0)
    qr = torch.einsum('bgci,cij->bgij', q, q_embedding)
    kr = torch.einsum('bgci,cij->bgij', k, k_embedding).transpose(2, 3)
    qk = torch.einsum('bgci, bgcj->bgij', q, k)


    # multiply by factors
    qr = torch.mul(qr, self.f_qr)
    kr = torch.mul(kr, self.f_kr)

    stacked_similarity = torch.cat([qk, qr, kr], dim=1)
    stacked_similarity = self.bn_similarity(stacked_similarity).view(N * W, 3, self.heads, H, H).sum(dim=1)
    #stacked_similarity = self.bn_qr(qr) + self.bn_kr(kr) + self.bn_qk(qk)
    # (N, heads, H, H, W)
    similarity = torch.softmax(stacked_similarity, dim=3)
    sv = torch.einsum('bgij,bgcj->bgci', similarity, v)
    sve = torch.einsum('bgij,cij->bgci', similarity, v_embedding)

    # multiply by factors
    sv = torch.mul(sv, self.f_sv)
    sve = torch.mul(sve, self.f_sve)

    stacked_output = torch.cat([sv, sve], dim=-1).view(N * W, self.out_channels * 2, H)
    output = self.bn_output(stacked_output).view(N, W, self.out_channels, 2, H).sum(dim=-2)
```

torch

```python
def call(self, x):
    if self.flag:
        x = tf.transpose(x, perm=[0, 1, 2, 3])
    else:
        x = tf.transpose(x, perm=[0, 2, 1, 3])
    N, W, H, C = x.shape
    x = tf.reshape(x, (-1, H, C)) # -1: N*W

    # Transformations
    x = self.to_qkv(x)
    x = tf.transpose(x, perm=[0, 2, 1])
    x = self.bn_qkv(x)

    qkv = tf.reshape(x, (-1, self.heads, self.dim_head_v * 2, H)) # -1: N*W
    q, k, v = tf.split(qkv, [self.dim_head_v // 2, self.dim_head_v // 2, self.dim_head_v], axis=2)

    # Calculate position embedding
    all_embeddings = tf.gather(self.relative, self.flatten_index, axis=1)
    all_embeddings = tf.reshape(all_embeddings, (self.dim_head_v * 2, self.dim, self.dim))
    q_embedding, k_embedding, v_embedding = tf.split(all_embeddings, [self.dim_head_qk, self.dim_head_qk, self.dim_head_v], axis=0)

    qr = tf.einsum('bgci,cij->bgij', q, q_embedding)
    kr = tf.transpose(tf.einsum('bgci,cij->bgij', k, k_embedding), perm=[0, 1, 3, 2])
    qk = tf.einsum('bgci, bgcj->bgij', q, k)

    # Multiply by factors
    qr = tf.multiply(qr, self.f_qr)
    kr = tf.multiply(kr, self.f_kr)

    stacked_similarity = tf.concat([qk, qr, kr], axis=1)
    stacked_similarity = self.bn_similarity(stacked_similarity)
    stacked_similarity = tf.reshape(stacked_similarity, (-1, 3, self.heads, H, H)) # -1: N*W
    stacked_similarity = tf.reduce_sum(stacked_similarity, axis=1)
    similarity = tf.nn.softmax(stacked_similarity, axis=2)

    sv = tf.einsum('bgij,bgcj->bgci', similarity, v)
    sve = tf.einsum('bgij,cij->bgci', similarity, v_embedding)

    # Multiply by factors
    sv = tf.multiply(sv, self.f_sv)
    sve = tf.multiply(sve, self.f_sve)

    stacked_output = tf.concat([sv, sve], axis=-1)
    stacked_output = self.bn_output(stacked_output)
    stacked_output = tf.reshape(stacked_output, (-1, W, self.out_channels, 2, H))
    output = tf.reduce_sum(stacked_output, axis=-2)
```

tf

展望

超低延迟交易系统

更智能的交易决策

多样化市场

数据隐私和安全

监管与合规

THANKS

软件正在重新定义世界

Software Is Redefining The World

欢迎探讨交流

E-mail: huangyicong@kaitong.tech